Pienin virittävä puu (minimum spanning tree)

Jatkossa "puu" tarkoittaa vapaata puuta (ks. s. 113) eli suuntaamatonta verkkoa, joka on

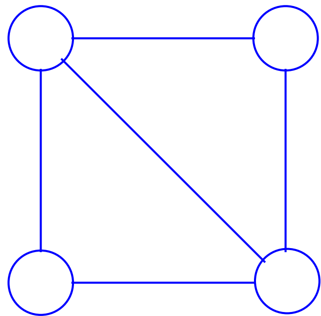
yhtenäinen: minkä tahansa kahden solmun välillä on polku

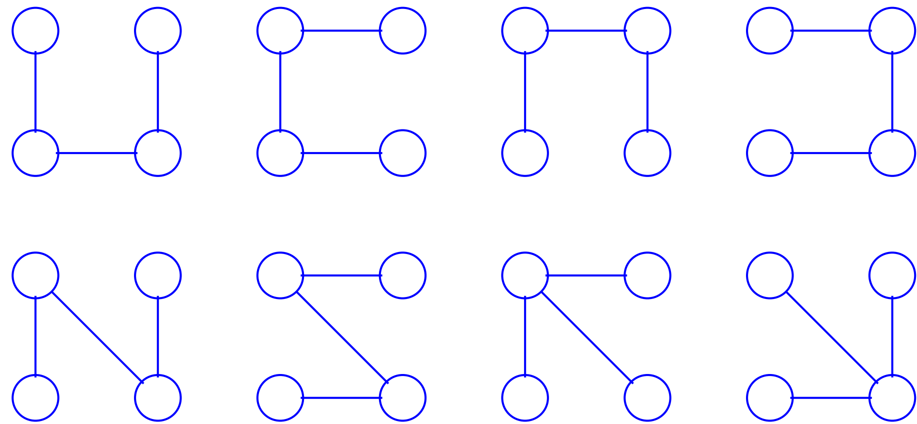
syklitön: minkä tahansa kahden solmun välillä on korkeintaan yksi yksinkertainen polku.

Yhtenäisen suuntaamattoman verkon $G = (V, E)$ virittävä puu on mikä tahansa puu (V, T), missä $T \subseteq E$. (Usein termiä "virittävä puu" käytetään myös pelkästä kaarijoukosta T.)

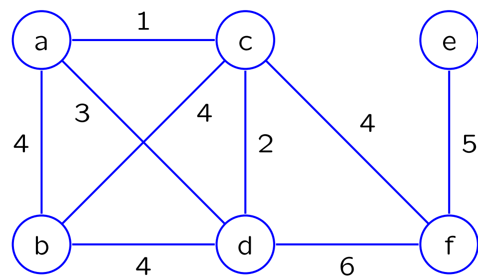
Siis virittävä puu saadaan valitsemalla joukosta E mahdollisimman vähän kaaria niin, että verkko pysyy yhtenäisenä. Virittävän puun kaarten lukumäärä on aina $|T| = |V| - 1$.

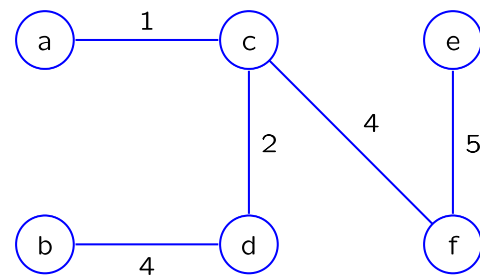
Pienin virittävä puu on virittävä puu, jonka paino $w(T) = \sum_{e \in T} w(e)$ on pienin mahdollinen.





Verkko ja sen kaikki virittävät puut





Painollinen verkko ja sen eräs pienin virittävä puu

Jatkossa esitettävät algoritmit pienimmän virittävän puun löytämiseksi noudattavat seuraavaa peruskaavaa:

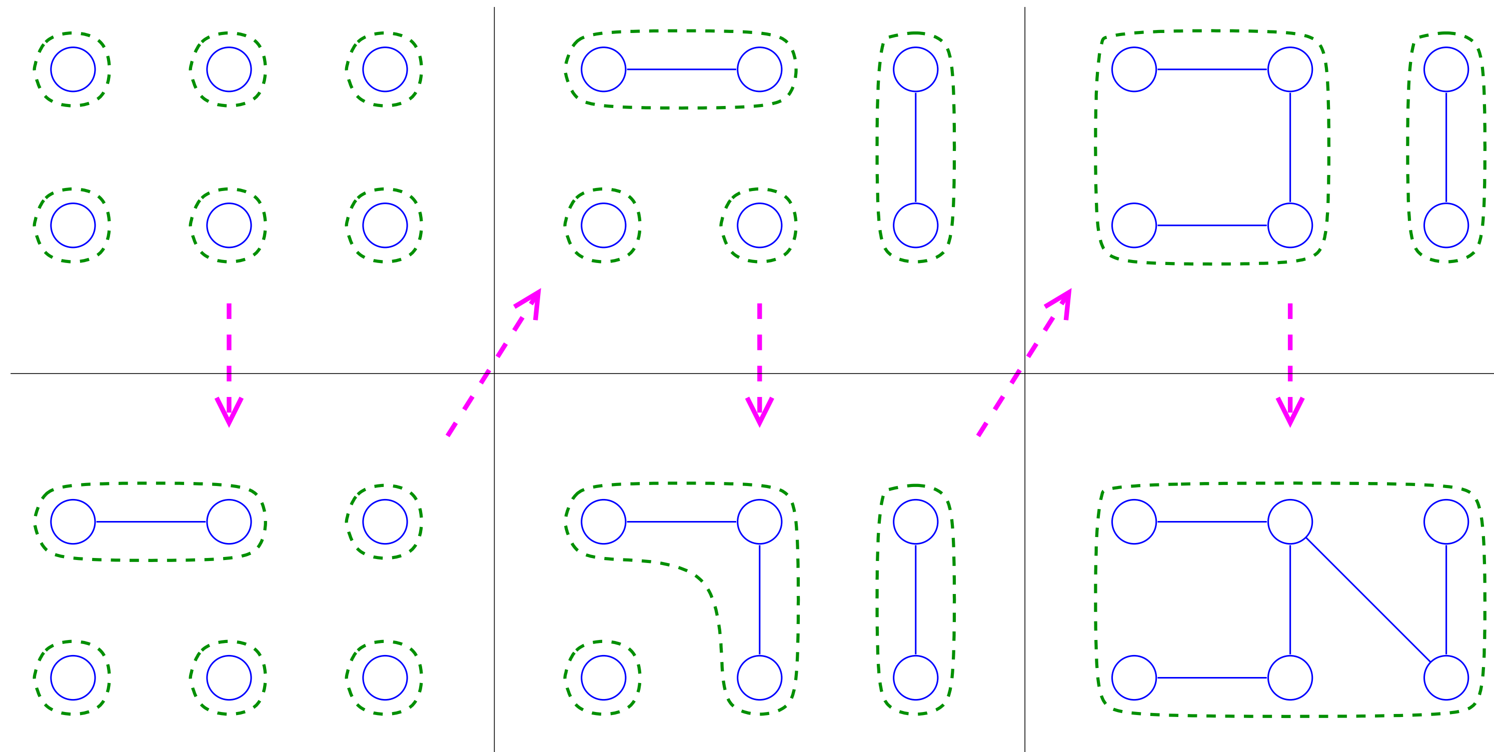
```
Spanning-Tree(G)
    F ← ∅
    while |F| < |V| − 1
        do valitse sellainen (u, v) ∈ E − F että verkossa
                                   (V, F ∪ {(u, v)}) ei ole kehää
           F ← F ∪ {(u, v)}
```

Kysymyksiä:

1. Mistä tiedämme, että sopiva kaari (u, v) on aina olemassa?

2. Miten kehän olemassaolo testataan tehokkaasti?

3. Miten (u, v) pitää valita, että saadaan pienin virittävä puu?



Eräs tapa muodostaa virittävä puu. Välivaiheina syntyneiden virittävien metsien komponentit (selitetään pian) ympäröity katkoviivalla.

Virittävä puu liittyy läheisesti verkon yhtenäisyyden käsitteeseen.

Olkoon $G = (V, E)$ suuntaamaton verkko, joka ei välttämättä ole yhtenäinen. Merkitään $u \sim v$, jos verkossa G on polku $u \rightsquigarrow v$. Relaatio $\sim$ on selvästi ekvivalenssirelaatio:

1. $u \sim u$ kaikilla u
2. $u \sim v$, jos ja vain jos $v \sim u$
3. jos $u \sim v$ ja $v \sim w$, niin $u \sim w$.

Voimme siis muodostaa sen ekvivalenssiluokat eli solmujoukot $V_1, \ldots, V_l \subseteq V$, missä

- jokaisella $u \in V$ on olemassa tasan yksi i, jolla $u \in V_i$
- jos $u, v \in V_i$, niin $u \sim v$
- jos $u \in V_i$ ja $v \in V_j$, missä $i \neq j$, niin $u \not\sim v$.

Näitä ekvivalenssiluokkia kutsutaan verkon yhtenäisiksi komponenteiksi, ja ne voidaan helposti muodostaa esim. syvyyssuuntaisella läpikäynnillä (mitä emme kuitenkaan tarvitse jatkossa).

(Vertaa suunnatun verkon vahvasti yhtenäisiin komponentteihin, s. 435.)

Kun verkon $G = (V, E)$ yhtenäiset komponentit V_i on annettu, voidaan osittaa myös $E = E_1 \cup \ldots \cup E_l$, missä luokkaan E_i tulevat komponentin V_i solmujen väliset kaaret. Määritelmän mukaan eri komponentteihin kuuluvien solmujen välillä ei ole kaaria. Verkko siis jakautuu osaverkkoihin $G_i = (V_i, E_i)$, jotka ovat yhtenäisiä.

Jos $F \subseteq E$ on sellainen, että verkko (V, F) on syklitön, sanomme verkkoa (V, F) verkon (V, E) virittäväksi metsäksi. Jos metsälle (V, F) muodostetaan yhtenäiset komponenttiverkot (V_i, F_i) ylläesitettyyn tapaan, kukin (V_i, F_i) on

- syklitön, koska $F_i \subset F$ ja (V, F) on syklitön
- yhtenäinen, koska kyseessä on yhtenäinen komponentti.

Siis kukin (V_i, F_i) on puu. Virittävä metsä on kokoelma puita

- jotka yhdessä kattavat koko verkon mutta
- joita ei ole yhdistetty toisiinsa.

Palataan nyt algoritmihahmotelmaan Spanning-Tree. Edellä esitettyä terminologiaa käyttäen se pitää yllä verkon (V, E) virittävää metsää (V, F).

Aluksi $F = \emptyset$ eli metsä käsittää $|\,V\,|$ yksisolmuista puuta.

Jokaisella askelella algoritmi lisää joukkoon F yhden kaaren (u, v), joka ei luo verkkoon (V, F) syklejä. Algoritmi siis pitää yllä invariantin, että (V, F) on virittävä metsä.

Lopuksi $|\,F\,| = |\,V\,| - 1$ eli metsässä on vain yksi $|\,V\,|$-solmuinen puu.

Väitämme, että Spanning-Tree ei voi joutua "umpikujaan", eli jos $|\,F\,| < |\,V\,| - 1$, on aina mahdollista valita sellainen $(u, v) \in E - F$, että $(V, F \cup \{\,(u, v)\,\})$ on syklitön.

Olkoon $|F| < |V| - 1$ ja (V, F) syklitön. Siis virittävässä metsässä (V, F) on ainakin kaksi puuta (V_1, F_1) ja (V_2, F_2). Olkoon $p \in V_1$ ja $q \in V_2$. Koska G on yhtenäinen, siinä on polku $p \rightsquigarrow q$. Olkoon (u, v) tämän polun ensimmäinen kaari, jolla $u \in V_1$ mutta $v \notin V_1$. Merkitään $F' = F \cup \{ (u, v) \}$. Väitämme, että (V, F') on syklitön.

Koska (V, F) on syklitön, niin mikä tahansa verkon (V, F') sykli kulkisi kaaren (u, v) kautta. Sykli voitaisiin siis kirjoittaa muotoon $(u, v, w_1, \ldots, w_t, u)$, missä kaaret (v, w_i), (w_t, u) ja (w_i, w_{i+1}) kaikilla $1 \leq i \leq t - 1$ kuuluvat joukkoon F. Siis $(v, w_1, \ldots, w_t, u)$ olisi polku $v \rightsquigarrow u$ verkossa (V, F). Tämä olisi vastoin oletusta, että u ja v ovat eri puissa. Siis verkossa (V, F') ei ole syklejä.

Olemme todenneet, että sovelias lisättävä kaari (u, v) on aina olemassa, kunnes virittävässä metsässä on vain yksi komponentti. Siis algoritmi tuottaa virittävän puun (V, F).

Tarkastellaan seuraavaksi tehokasta testausmenetelmää algoritmin Spanning-Tree ehdolle

verkko $(V, F \cup \{ (u,v) \})$ on syklitön,

missä siis (V, F) voidaan olettaa syklittömäksi.

Edellä esitetyn perusteella jos u ja v kuuluvat metsässä (V, F) eri puihin, niin $(V, F \cup \{ (u,v) \})$ on syklitön.

Toisaalta jos solmut u ja v kuuluvat samaan puuhun ja $u \neq v$, niiden välillä on polku $(u, w_1, \ldots, w_t, v)$, jonka kaaret kuuluvat joukkoon F. Siis verkossa $(V, F \cup \{ (u,v) \})$ on sykli $(u, w_1, \ldots, w_t, v, u)$.

Olemme todenneet, että ylläoleva ehto voidaan kirjoittaa muotoon

solmut u ja v ovat metsässä (V, F) eri puissa.

Tämän voi testata esim. syvyyssuuntaisella läpikäynnillä ajassa $O(|V| + |E|)$. Näitä testejä joudutaan kuitenkin tekemään hyvin paljon. Suoritusta voi oleellisesti nopeuttaa pitämällä yllä sopivaa tietorakennetta.

Erillisten joukkojen yhdisteet

Esittelemme tietorakenteen, josta on hyötyä yleisemminkin kuin virittävän metsän komponenttien ylläpidossa.

On annettu perusjoukko X ja kokoelma $\mathcal{S}$ sen erillisiä osajoukkoja. Siis $\mathcal{S} = \{ S_1, \ldots, S_n \}$, missä $S_i \subseteq X$ ja $S_i \cap S_j = \emptyset$, jos $i \neq j$. Haluamme tukea seuraavanlaisia operaatioita:

- Union(S_i, S_j): Kokoelmaan $\mathcal{S}$ liitetään uusi joukko, joka saa alkioikseen joukkojen S_i ja S_j alkiot. Vanhat joukot S_i ja S_j poistuvat kokoelmasta.
- Find(x): palauttaa sen joukon S_i tunnuksen, johon x kuuluu.

Täsmennämme tätä niin, että joukon S_i tunnuksena on jokin sen alkio $x \in S_i$. Joukon tunnus voi ajan myötä vaihtua, mutta annetulla ajanhetkellä se on yksikäsitteinen.

Saamme seuraavat operaatiot:

Make-Set(x): luo uuden joukon $\{x\}$, jonka tunnukseksi tulee x.
Union(x, y): yhdistää joukot, joiden tunnukset ovat x ja y.
Find(x): palauttaa sen joukon tunnuksen, johon x kuuluu.

Siis erityisesti operaation Union(Find(x), Find(y)) jälkeen kutsujen Find(x) ja Find(y) pitää palauttaa sama alkio. Koska mikään operaatio ei pilko joukkoja, tämä pätee jatkossa aina.

Esimerkki 7.10:

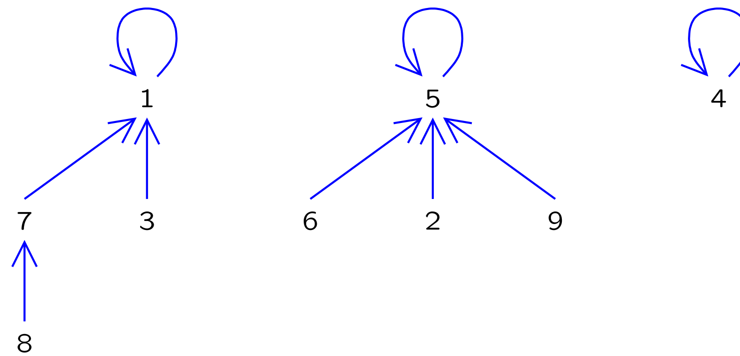
operaatio	joukot
Make-Set(1), ... Make-Set(9)	$\{\underline{1}\}\ \{\underline{2}\}\ \{\underline{3}\}\ \{\underline{4}\}\ \{\underline{5}\}\ \{\underline{6}\}\ \{\underline{7}\}\ \{\underline{8}\}\ \{\underline{9}\}$
Union(1, 3)	$\{\underline{1},3\}\ \{\underline{2}\}\ \{\underline{4}\}\ \{\underline{5}\}\ \{\underline{6}\}\ \{\underline{7}\}\ \{\underline{8}\}\ \{\underline{9}\}$
Union(2, 5)	$\{\underline{1},3\}\ \{2,\underline{5}\}\ \{\underline{4}\}\ \{\underline{6}\}\ \{\underline{7}\}\ \{\underline{8}\}\ \{\underline{9}\}$
Union(6, 5)	$\{\underline{1},3\}\ \{2,\underline{5},6\}\ \{\underline{4}\}\ \{\underline{7}\}\ \{\underline{8}\}\ \{\underline{9}\}$
Find(7)	palauttaa 7
Find(6)	palauttaa 5
Union(7, 8)	$\{\underline{1},3\}\ \{2,\underline{5},6\}\ \{\underline{4}\}\ \{\underline{7},8\}\ \{\underline{9}\}$
Union(5, 9)	$\{\underline{1},3\}\ \{2,\underline{5},6,9\}\ \{\underline{4}\}\ \{\underline{7},8\}$
Union(7, 1)	$\{\underline{1},3,7,8\}\ \{2,\underline{5},6,9\}\ \{\underline{4}\}$

Joukkojen edustajat alleviivattu (mutta olisi voitu valita toisinkin). □

Toteutamme erilliset joukot puurakenteena:

- Jokainen joukko esitetään juurellisena puuna, jonka solmuissa on joukon alkiot.
- Jokaiseen alkiooon x liittyy osoitin $p[x]$:
  - Jos x on puun juuri (mukaanluettuna yksisolmuinen puu), niin $p[x] = x$.
  - Muuten $p[x]$ osoittaa solmun x vanhempaa puussa.
- Puun juuressa on joukon tunnusalkio.

Esimerkki 7.11: Joukkojen $\{\underline{1}, 3, 7, 8\}$, $\{2, \underline{5}, 6, 9\}$ ja $\{\underline{4}\}$ puuesitys.



□

Jos tehokkuudesta ei välitetä, operaatiot on suoraviivaista toteuttaa:

Make-Set-0.1(x)

$p[x] \leftarrow x$

Union-0.1(x, y)

$p[y] \leftarrow x$

Find-0.1(x)

```
while p[x] ≠ x
      do x ← p[x]
return x
```

Operaatiot Make-Set ja Union sujuvat selvästi vakioajassa, mutta Find vie hakupolun pituuden suhteen lineaarisen ajan. Kahdella helpolla optimoinnilla polut lyhenevät oleellisesti:

tasapainotus Union-operaatiossa
poluntiivistys Find-operaatiossa.

Tasapainottavan Union-toteutuksen idea on liittää jokaiseen solmuun x luokitus $rank[x]$, joka kertoo pisimmän solmuun x johtavan polun pituuden (kun mahdollisia silmukoita $x \to x$ ei lasketa).

Siis suurin luokitus $\max_x rank[x]$ on yläraja Find-operaation vaativuudelle.

Alustus tulee muotoon

Make-Set(x)

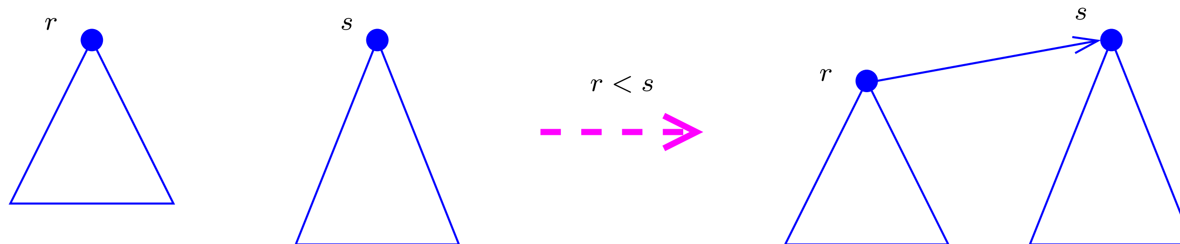
$p[x] \leftarrow x$
$rank[x] \leftarrow 0$

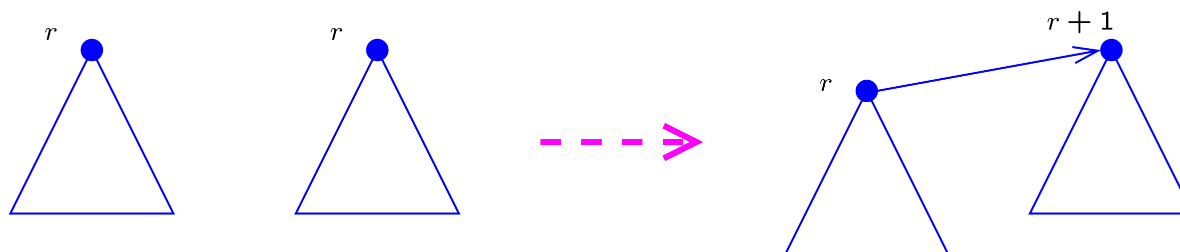
Jotta Union-operaatiot sujuisivat nopeasti, halutaan pitää $rank$-arvot mahdollisimman pienenä. Kun sijoitetaan $p[y] \leftarrow x$, niin luokitusta $rank[x]$ pitää päivittää seuraavasti:

$$rank[x] \leftarrow \max \{ rank[x], rank[y] + 1 \} .$$

Jos $rank[y] < rank[x]$, niin luokitukset eivät kasva! Siis kannattaa sijoittaa pienempi luokitus suuremman luokituksen alipuuksi:

```
Union(x, y)
    if rank[y] < rank[x]
        then p[y] ← x
    elseif rank[x] < rank[y]
        then p[x] ← y
    else
            p[y] ← x
            rank[x] ← rank[x] + 1
```



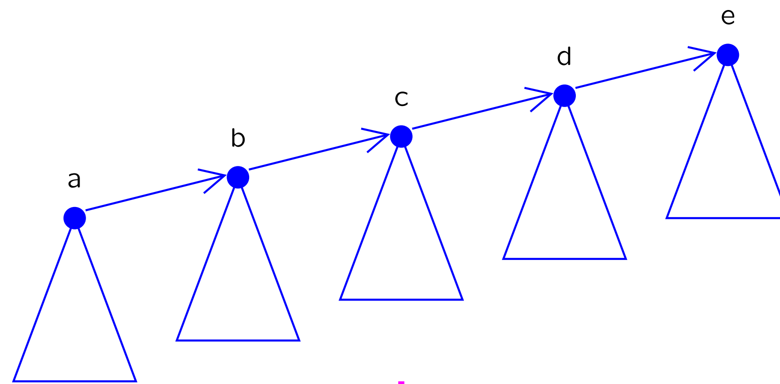


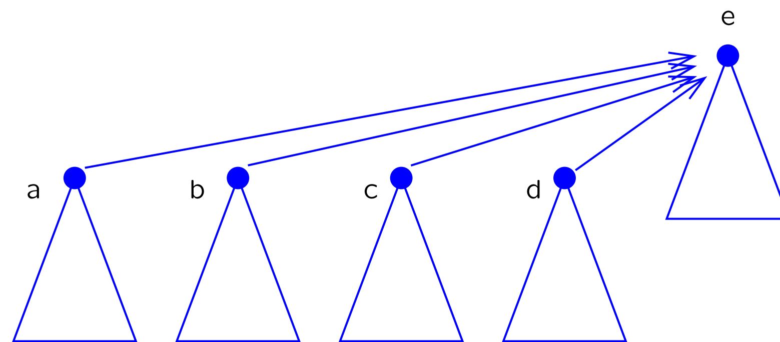
Union ja tasapainotus $rank$-arvoilla.

Induktiolla nähdään helposti, että jos $rank[x] = k$, niin solmulla x on ainakin 2^k jälkeläistä puussa.

Siis jos alkioita kaikkiaan on n, niin kaikki polunpituudet ovat $O(\log n)$. Mikä tahansa m Union-Find-operaation jono voidaan suorittaa ajassa $O(m \log n)$.

Tietorakennetta voidaan vielä tehostaa suorittamalla Find-operaation yhteydessä poluntiivistys: Kun on löydetty polku $x \to p[x] \to p[p[x]] \to \ldots \to y$, missä y on puun juuri, niin oikaistaan kaikki osoittimet $p[x]$, $p[p[x]]$ jne. osoittamaan suoraan juureen y. Tällöin seuraavat Find-operaatiot nopeutuvat.





Find ja poluntiivistys

Saadaan seuraava algoritmi:

```
Find(x)
    z ← x
    while p[z] ≠ z
        do z ← p[z]
    y ← z
    z ← x
    while p[z] ≠ z
        do z, p[z] ← p[z], y
    return y
```

Huomaa, että $rank$-arvoja ei Find-operaatiossa päivitetä, joten ne menettävät tarkan vastaavuutensa polunpituuksiin.

Tasapainotusta ja poluntiivistystä käytettäessä voidaan osoittaa, että m Union-Find-operaatiota n alkion perusjoukossa vie vain ajan $O(m\alpha(n))$, missä α on eräs *erittäin* hitaasti kasvava funktio: karkeasti arvioiden $\alpha(n) \leq 4$ kun $n \leq 10^{80}$. (Todistus sivuutetaan, ks. esim. Cormen et al. luku 21.)

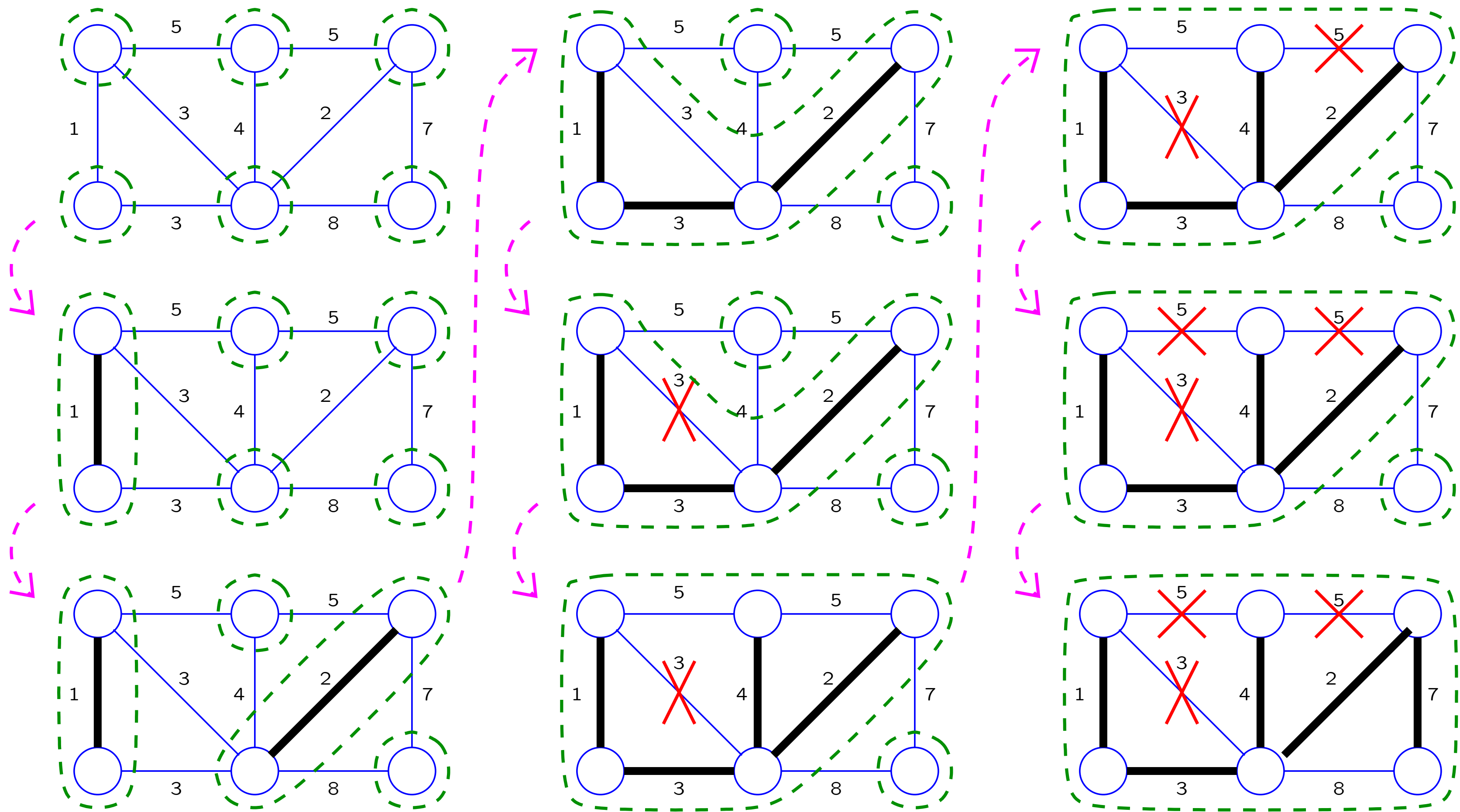
Sama asymptoottinen aikavaativuus saadaan muillakin samanhenkisillä tasapainotus- ja polunoikaisuheuristiikoilla.

Kruskalin algoritmi

Meillä on nyt tarvittavat komponentit pienimmän virittävän puun tehokkaaseen löytämiseen. Kruskalin algoritmissa virittävän puun perusalgoritmiin lisätään yksinkertainen ahne heuristiikka: kokeillaan aina painoltaan pienintä jäljellä olevaa kaarta.

```
Kruskal(G)
    F ← ∅
    for kaikille v ∈ V
        do Make-Set(v)
    Järjestä E kaaren painon mukaan kasvavasti.
    for kaikille (u, v) ∈ E painon mukaan kasvavassa järjestyksessä
        do if Find(u) ≠ Find(v)
            then F ← F ∪ { (u, v) }
                Union(Find(u), Find(v))
    return (V, F)
```

Jos usealla kaarella on sama paino, ne voidaan käsitellä missä tahansa järjestyksessä.



Esimerkki Kruskalin algoritmin toiminnasta.

Tarkastellaan seuraavaksi Kruskalin algoritmin aikavaativuutta:

- Make-Set-operaatiot $O(|V|)$
- muut Union- ja Find-operaatiot yhteensä $O(|E|\alpha(|V|))$
- kaarten järjestäminen $O(|E|\log|E|)$
- muu kirjanpito yms. $O(|E|)$.

Siis aikavaativuutta dominoi kaarten järjestämisen $O(|E|\log|E|)$.

Usein virittävä puu tulee valmiiksi, ennen kuin kaikki kaaret on käsitelty. Tällaisissa tapauksissa algoritmia voidaan jonkin verran tehostaa käyttämällä kekoa sen sijaan, että heti järjestetään kaikki kaaret:

```
Kruskal-with-Heap(G)
    F ← ∅
    for kaikille v ∈ V
        do Make-Set(v)
    H ← Build-Heap(E)
    while |F| < |V| − 1
        do (u, v) ← Heap-Delete-Min(H)
           if Find(u) ≠ Find(v)
             then F ← F ∪ { (u, v) }
                  Union(Find(u), Find(v))
    return (V, F)
```

Asymptoottinen pahimman tapauksen aikavaativuus on kuitenkin sama kuin ennenkin.

Kruskalin algoritmi oikeellisuus ei ole aivan ilmeinen. Todetaan ensin, että se palauttaa jonkin virittävän puun (V, F):

Kaaren (u, v) sisällyttäminen joukkoon F aiheuttaa yhdistämisen $\mathsf{Union}(\mathsf{Find}(u), \mathsf{Find}(v))$. Siis ehto $\mathsf{Find}(u) \neq \mathsf{Find}(v)$ on tosi vain, jos u ja v ovat verkon (V, F) eri komponenteissa. Aiemmin esitetyn perusteella (V, F) pysyy syklittömänä. Siis algoritmin palauttama (V, F) on virittävä metsä.

Oletuksen mukaan G on yhtenäinen. Jos (V, F) algoritmi suorituksen päättyessä ei olisi yhtenäinen, niin verkossa G olisi jokin kahta metsän (V, F) puuta yhdistävä kaari e, jota ei ole otettu mukaan joukkoon F. Koska puut eivät suorituksen aikana lisäänny, e yhdisti kahta eri puuta jo silloin, kun e oli kokeiltavana, jolloin se olisi pitänyt ottaa mukaan joukkoon F; ristiriita.

Siis suorituksen päättyessä (V, F) on virittävä puu.

Seuraavan tuloksen perusteella voidaan osoittaa, että algoritmin palauttama virittävä puu on pienin:

Lause 7.12: Olkoon $U \subset V$, $\overline{U} = V - U$ ja $(u, v) \in E$ painoltaan pienin sellainen kaari, jonka toinen päätepiste on joukossa U ja toinen joukossa $\overline{U}$.

Olkoon lisäksi F sellainen joukko kaaria, että

- jos $(r, s) \in F$, niin r ja s ovat joko kumpikin joukossa U tai kumpikin joukossa $\overline{U}$
- $F \subseteq T$ jollekin pienimmälle virittävälle puulle (V, T).

Nyt $F \cup \{ (u, v) \} \subseteq T'$ jollekin pienimmälle virittävälle puulle (V, T').

Lauseen 7.12 avulla nähdään, että algoritmi pitää yllä invarianttia

$F \subseteq T$ jollakin pienimmällä virittävällä puulla (V, T).

Aluksi $F = \emptyset$, joten invariantti pätee triviaalisti.

Kun joukkoon F ollaan lisäämässä kaarta (u, v), valitaan joukkoon U ne solmut, jotka ovat (ennen lisäystä) solmun u kanssa samassa puussa. Joukko U toteuttaa lauseen ehdot, joten lauseen nojalla $F \cup \{ (u, v) \}$ toteuttaa invariantin.

Lopuksi (V, F) on virittävä puu, joten invariantin nojalla se on pienin sellainen, kuten haluttiin.

Lauseen 7.12 todistus: Olkoon (V, T) pienin virittävä puu, jolla $F \subseteq T$. Jos $(u, v) \in T$, väite on selvä.

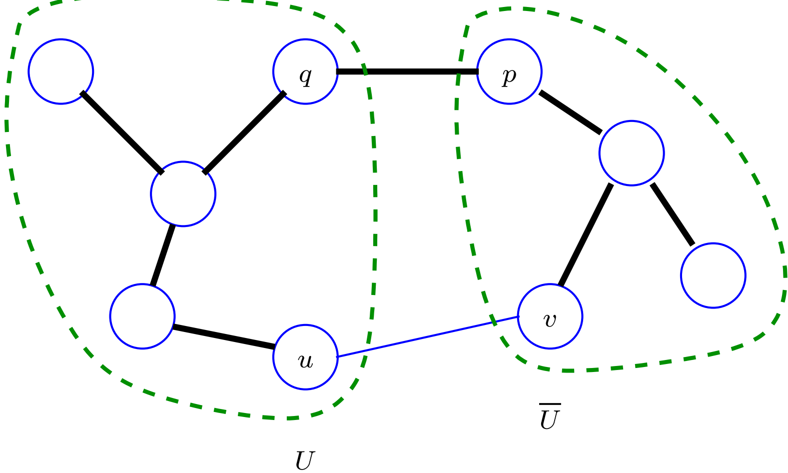
Oletetaan $(u, v) \notin T$. Verkossa $(V, T \cup \{ (u, v) \})$ on kehä, joka sisältää kaaren (u, v) ja sen lisäksi ainakin yhden toisen kaaren (p, q), jonka toinen päätepiste on joukossa U ja toinen joukossa $\overline{U}$. Erityisesti $(p, q) \notin F$. Valitaan $T' = T \cup \{ (u, v) \} - \{ (p, q) \}$.

Kaaren (p, q) poistaminen jakaa puun (V, T) kahteen komponenttiin, jotka (u, v) taas yhdistää. Siis (V, T') on virittävä puu.

Oletuksen mukaan $w(u, v) \leq w(p, q)$, joten

$$\sum_{e \in T'} w(e) = \sum_{e \in T} w(e) - w(p, q) + w(u, v) \leq \sum_{e \in T} w(e).$$

Koska (V, T) on pienin virittävä puu, niin myös (V, T') on. $\square$


q
p
u
v
U
$\overline{U}$

Primin algoritmi

Perusidea on samantapainen kuin Kruskalin algoritmissa. Nyt kuitenkin pidetään yllä vain yhtä puuta ja lisätään siihen kaaria yksi kerrallaan.

Toisin sanoen algoritmi pitää yllä puuta (S,T), missä $S \subseteq V$ ja $T \subseteq E$. Kuten Kruskalin algoritmissa, nytkin pidetään yllä invarianttia

> jollakin pienimmällä virittävällä puulla (V,T') pätee $T \subseteq T'$.

Puuhun lisätään seuraavaksi aina se solmu $v \in V - S$, jonka etäisyys

$$key[v] = \min_{u \in S} w(u,v)$$

on pienin. Solmut pidetään tämän avaimen mukaisessa prioriteettijonossa.

Kun v liitetään joukkoon S, samalla joukkoon T liitetään (u,v), missä $u \in S$ on sellainen, että $w(u,v) = key[v]$.

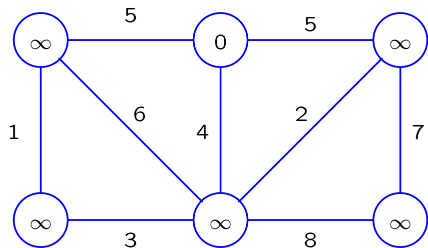
Algoritmi ei pidä eksplisiittisesti kirjaa joukoista S ja T.

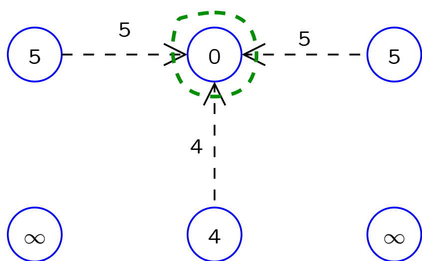
Joukon S solmuja ovat ne, jotka eivät enää ole prioriteettijonossa Q.

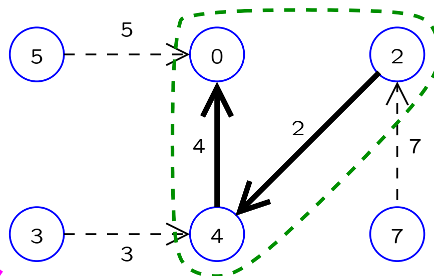
Jokaiseen solmuun $v \in V - S$ liitetään osoitin $p[v]$, joka kertoo sitä lähinnä olevan joukon S solmun. Kun v liitetään joukkoon S, arvo $p[v]$ jäädytetään. Lopulta puuhun T tulee kaikki kaaret $(v, p[v])$, missä $v \neq r$.

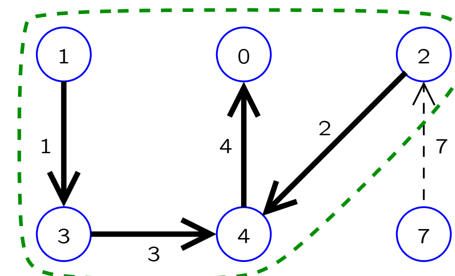
Annamme algoritmille parametrina solmun r, joka toimii puun "siemenenä".

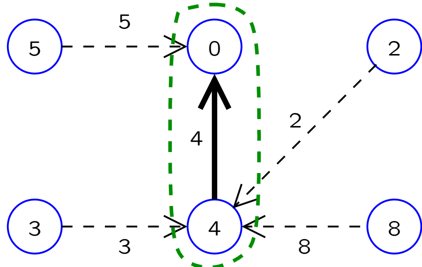
```
Prim(G, w, r)
    for jokaiselle v ∈ V do key[v] ← ∞
    key[r] ← 0
    for jokaiselle v ∈ V do Insert(Q, r, key[r])
    while Q ≠ ∅
        do u ← Delete-Min(Q)
           for jokaiselle v ∈ Adj[u]
               do if v ∈ Q and key[v] > w(u, v)
                      then key[v] ← w(u, v)
                           Decrease-Key(Q, v, w(u, v))
                           p[v] ← u
```

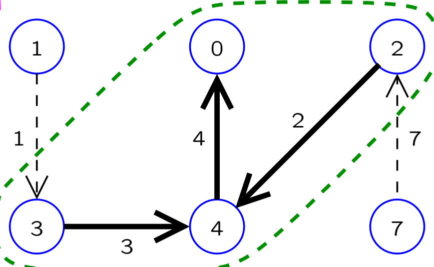


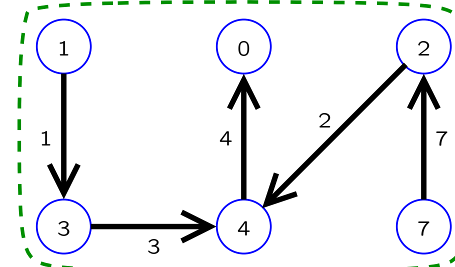












Esimerkki Primin algoritmin toiminnasta.

Algoritmista nähdään suoraan, että arvot $key[v]$ ja $p[v]$ ovat kuten edellä sanottiin:

$$\begin{aligned} key[v] &= \min_{u \in S} w(u,v) \qquad \text{kun } v \notin S \\ w(p[v],v) &= key[v], \end{aligned}$$

missä $S = V - Q$.

Lisäksi kun määritellään $T = \{ (p[v], v) \mid v \in S - \{ r \} \}$, niin verkko (S,T) on syklitön, sillä joukkoon T ei koskaan lisätä kaaria joukossa S jo olevien solmujen välille.

Algoritmin oikeellisuuden ei-trivaali osuus on osoittaa invariantti

> jollekin pienimmälle virittävälle puulle (V,T') pätee $T \subset T'$.

Tähän käytämme jälleen lausetta 7.12 (s. 514).

Olkoon S ja T kuten edellä. Tarkastellaan algoritmia, kun ollaan lisäämässä $S \leftarrow S \cup \{ u \}$. Tehdään induktio-oletus, että $T \subseteq T'$ jollain pienimmällä virittävällä puulla (V, T').

Kaikilla $v \in V - S$ kaari $(p[v], v)$ on painoltaan pienin kaari joukosta S solmuun v, ja kaaren paino on $key[v]$. Algoritmi valitseen sen u, jolla avain $key[u]$ on pienin. Siis $(p[u], v)$ on painoltaan pienin kaari joukosta S joukkoon $V - S$. Koska joukossa T ei ennestään ole kaaria joukosta S joukkoon $V - S$, lauseen 7.12 ehdot pätevät. Siis $T \cup \{ (p[u], u) \} \subseteq T''$ jollekin pienimmälle virittävälle puulle (V, T'').

Siis invariantti pätee. Lopuksi $S = V$ ja (S, T) on koko verkon pienin virittävä puu.

Primin algoritmin ensimmäinen for-silmukka vie ajan $O(|V|)$.

Jos prioriteettijono toteutetaan kekona, sen alustaminen voidaan suorittaa ajassa $O(|V|)$ käyttämällä algoritmia Build-Heap.

Delete-Min-operaatioita suoritetaan $|V|$ ja Decrease-Key-operaatioita korkeintaan $|E|$ kappaletta. Kekototeutuksella kukin näistä menee ajassa $O(\log|V|)$. Siis while-silmukan ja samalla koko algoritmin aikavaativuudeksi tulee $O((|V|+|E|)\log|V|)$ eli sama kuin Kruskalin algoritmilla.

Kuten Dijkstran algoritmissa, tässäkin asymptoottista aikavaativuutta voidaan parantaa toteuttamalla prioriteettijono Fibonacci-kekona. Tällöin kutakin Decrease-Key-operaatiota kohden tarvitaan vain vakioaika. Kokonaisaikavaativuudeksi tulee $O(|E|+|V|\log|V|)$.

Jos verkko on tiheä eli $|E| = \Theta(|V|^2)$, Primin algoritmi Fibonacci-keolla toteutettuna saa aikavaativuuden $O(|E| + |V| \log |V|) = O(|V|^2)$. Sama aikavaativuus saadaan helpomminkin:

Prim-Dense(G, w, r)

```
key[r] ← 0
S ← { r }
for jokaiselle v ∈ V − { r }
      do key[v] ← w(r, v)
         p[v] ← r
while S ≠ V
      do valitse u, jolla key[u] on pienin
         S ← S ∪ { u }
         for kaikille v ∈ V
               do if v ∉ S and w(u, v) < key[v]
                     then key[v] ← w(u, v)
                          p[v] ← u
```

Siis Primin algoritmi aikavaativuus tiheille verkoille on $O(|V|^2)$ (myös ilman Fibonacci-kekoa, jonka vakiokertoimet ovat suuret).

Tiheillä verkoilla Primin algoritmin aikavaativuus $O(|V|^2)$ on parempi kuin Kruskalin $O(|E|\log|E|) = O(|V|^2\log|V|)$.

Harvoilla verkoilla Kruskalin pahimman tapauksen aikavaativuus $O(|E|\log|V|)$ on sama kuin Primin (jos unohdetaan Fibonacci-keko).

Monissa tilanteissa Kruskal on kuitenkin parempi, kuin pahimman tapauksen analyysi antaa ymmärtää:

- jos pienimpään virittävään puuhun tulevat kaaret ovat kaikki hyvin kevyitä, algoritmi pysähtyy, ennen kuin kaikkia kaaria on tarvinnut käydä läpi

- jos painot ovat pieniä kokonaislukuja, järjestäminen sujuu nopeammin kuin $\Omega(|E|\log|E|)$.

Siis algoritmien keskinäinen paremmuus on tapauskohtaista.

Esimerkki 7.13: Halutaan osittaa verkon $G = (V, E)$ solmut kahteen luokkaan V_1 ja V_2 siten, että luokkien välinen pienin etäisyys

$$d(V_1, V_2) = \min \{ w(u, v) \mid u \in V_1, v \in V_2 \}$$

on mahdollisimman suuri. (Muistetaan, että osituksessa $V_1 \cap V_2 = \emptyset$ ja $V_1 \cup V_2 = V$.)

Intuitiivinen tulkinta on, että

- $w(u, v)$ on jokin mitta alkioiden u ja v erilaisuudelle
- alkiot halutaan jakaa kahteen mahdollisimman selvästi toisistaan erottuvaan luokkaan.

Annettu ongelma on erikoistapaus ryvästämisestä, joka on tärkeä menetelmä data-analyysissä. Halutaan osittaa perusjoukko X rypäisiin $X_1, \ldots, X_k$ siten, että

- eri rypääseen kuuluvilla u ja v erilaisuus $d(u, v)$ on mahdollisimman suuri ja
- samaan rypääseen kuuluvilla u ja v erilaisuus $d(u, v)$ on mahdollisimman pieni.

Tavoite voidaan täsmentää eri tavoin, mutta yleensä ongelma on laskennallisesti hankala. Tässä tarkasteltu erikoistapaus ratkeaa kuitenkin helposti muodostamalla pienin virittävä puu.

Väitämme, että ongelma ratkeaa seuraavalla algoritmilla:

1. Muodosta verkolle pienin virittävä puu (V, T).

2. Olkoon e joukon T painoltaan suurin kaari.

3. Valitse luokiksi V_1 ja V_2 metsän $(V, T - \{ e \})$ yhtenäiset komponentit.

Tulos on sama, kuin jos ajettaisiin Kruskalin algoritmia, mutta lopetettaisiin juuri ennen viimeisen kaaren lisäämistä virittävään puuhun.

Todetaan ensin, että muodostettujen luokkien V_1 ja V_2 pienin etäisyys on

$$\min \{ w(u,v) \mid u \in V_1, v \in V_2 \} = w(e).$$

Valitaan mitkä tahansa $u \in V_1$ ja $v \in V_2$. Nyt $T' = (T - \{ e \}) \cup \{ (u,v) \}$ on virittävä puu, jonka paino on

$$w(T') = w(T) - w(e) + w(u,v).$$

Koska T on pienin virittävä puu, on oltava $w(u,v) \geq w(e)$.

Siis e on luokkia V_1 ja V_2 yhdistävistä kaarista painoltaan pienin.

Todetaan sitten, että millä tahansa solmujen osituksella (U_1, U_2) pätee

$$\min\{\, w(u,v) \mid u \in U_1, v \in U_2 \,\} \le w(e).$$

Tämä seuraa suoraan siitä, että ainakin yksi puun T kaari yhdistää joukkoja U_1 ja U_2, ja e on puun T kaarista painoltaan suurin.

Edellisistä kahdesta toteamuksesta seuraa, että algoritmin tuottama ositus (V_1, V_2) maksimoi lyhimmän etäisyyden

$$\min\{\, w(u,v) \mid u \in V_1, v \in V_2 \,\}.$$

□