Lause 8.4.3. *Läpikäydään (suunnattu tai suuntaamaton) verkko G leveyssuunnassa lähtien alkusolmusta $s \in V(G)$. Mielivaltaiselle solmulle $v \in V(G)$ pätee:*

1. *Läpikäynti tavoittaa solmun v*

   jos ja vain jos

   v on saavutettavissa solmusta s.

2. *Läpikäynnin päätteeksi $v.\text{depth} = \delta(s, v)$.*

3. *Jos $v \neq s$ on saavutettavissa alkusolmusta s, niin eräs vähäkaarisin polku on*

   $$\underbrace{s \to \cdots \to v.\pi}_{\mathcal{P}} \to v$$

   missä $\mathcal{P}$ on mielivaltainen vähäkaarisin polku alkusolmusta s solmuun $v.\pi$.

Todistus. Tehdään *vastaoletus* väitteeseen 2: Jokin solmu $v \in V(G)$ saakin väärän arvon $v.\text{depth} \neq \delta(s, v)$.

1. Voidaan valita sellainen v, että $\delta(s, v)$ on mahdollisimman pieni.

2. Selvästi $v \neq s$.

3. Huomion 4 nojalla $v.\text{depth} > \delta(s, v)$.

4. Solmu v on saavutettavissa solmusta s, koska muuten $\delta(s, v) = +\infty$, vastoin päätelmää 3.

5. Tarkastellaan mielivaltaista vähäkaarisinta saavuttavaa polkua

   $$\overbrace{\underbrace{s \to \quad \cdots \quad \to u}_{\delta(s,u) \text{ kaarta}} \to v}^{\delta(s,v) = \delta(s,u) + 1 \text{ kaarta}}$$

   jollaisia on olemassa päätelmän 4 nojalla.

6. Nyt saadaan epäyhtälö

   $$\begin{aligned} v.\text{depth} &> \delta(s, v) \quad \text{(päätelmä 3)} \\ &= \delta(s, u) + 1 \quad \text{(polku 5)} \\ &= u.\text{depth} + 1 \\ &\qquad \text{(valinta 1 ja polku 5).} \end{aligned}$$

7. Mikä voisi solmun v väri olla sillä hetkellä kun solmu u poistuu työjonosta Q?

   Valkea? Silloin suoritettaisiin rivi 16 solmulle v, ja saataisiin ristiriita päätelmän 6 kanssa.

   Musta? Silloin solmu v olisi jo poistunut työjonosta Q, ja huomio 6 olisi ristiriidassa päätelmän 6 kanssa.

   Harmaa? Silloin solmu v olisi harmaantunut poistettaessa työjonosta Q jotakin aikaisempaa solmua w kuin u.

   - Silloin $v.\text{depth} = w.\text{depth} + 1$ riviltä 16.
   - Huomion 6 nojalla $w.\text{depth} \leq u.\text{depth}$.

   Taas ristiriita päätelmän 6 kanssa.

Siis lauseen väite 2 on todistettu.

- Solmu v on saavutettavissa solmusta s

  jos ja vain jos

  $\delta(s, v) < +\infty$

  jos ja vain jos

  $v.\text{depth} < +\infty$ (lauseen väitteen 2 nojalla)

  jos ja vain jos

  syvyyssuuntainen läpikäynti käy muuttamassa kenttää $v.\text{depth}$ sen rivillä 3 asetetusta alkuarvosta $+\infty$.

  Siis myös lauseen väite 1 on todistettu.

- Kun $v.\pi \neq \text{NULL}$, pätee

  $$v.\text{depth} = v.\pi.\text{depth} + 1$$

  rivin 17 ja huomion 2 nojalla.

  Siis myös lauseen väite 3 seuraa väitteestä 2. □

8.5 Lyhyimmät polut

- Tarkastellaan seuraavaa ongelmaa:
  - Syötteenä annetaan
    * *numeroilla painotettu* verkko G (joka voi olla suuntaamaton tai suunnattu)
    * alkusolmu $s \in V(G)$.
  - Kysytään jokaiselle solmulle $u \in V(G)$
    * lyhyimmän polun pituus $\delta(s,u)$ solmusta s solmuun u kalvojen 8.1 määritelmän mukaisesti
    * laskettuna kenttään $u.\mathrm{dist}$
    * ja jokin vastaava polku
      $$u \xleftarrow{w_1} u.\pi \xleftarrow{w_2} u.\pi.\pi \xleftarrow{w_3} \cdots \xleftarrow{w_k} s$$
      missä siis
      $$w_1 + w_2 + w_3 + \cdots + w_k = \delta(s,u)$$
      laskettuna kenttään $u.\pi$.

- Kalvoilla 8.4.2 ratkaistiin sama ongelma *painottamattomilla* verkoilla
  - eli kun kaikki kaaripainot ovat $= 1$
  - eli kun kaikki kaaripainot ovat sama $c \geq 0$.
- *Negatiiviset* kaaripainot mutkistavatkin tilannetta:
  - Sallitaanko suunnatussa verkossa G *sykli*
    $$p_0 \xrightarrow{w_1} p_1 \xrightarrow{w_2} p_2 \xrightarrow{w_3} \cdots \xrightarrow{w_k} p_k = p_0$$
    jonka kokonaispituus
    $$w_1 + w_2 + w_3 + \cdots + w_k < 0?$$
    Eli solmu $p_0 \in V(G)$ jolla
    $$\delta(p_0, p_0) < 0?$$
    Silloin $\delta(u,v) = -\infty$ aina kun on polku
    $$u \xrightarrow{w_1} \cdots \xrightarrow{w_k} p_0 \xrightarrow{w_{k+1}} \cdots \xrightarrow{w_\ell} v.$$
  - Sallitaanko edes *yksittäiset* negatiiviset kaaripainot, kunhan niistä ei muodostu negatiivista sykliä?

Dijkstran * algoritmi kalvoilla 8.5.2 ratkaisee tehtävän

- kalvojen 8.3 vieruslistoilla
- jos negatiivisia kaaria ei sallita lainkaan.

Floydin algoritmi kalvoilla 8.5.3 ratkaisee tehtävän

- yleisemmän muodon jossa kysytään lyhyintä polkua *kaikkien* solmuparien $u, v \in V(G)$ välillä
- kalvojen 8.3 vierusmatriiseilla
- jos sallitaan negatiiviset kaaret muttei syklejä.

Bellmanin ja Fordin algoritmi

- ratkaisee tehtävän perusmuodon
- huomaa negatiivisen syklin olemassaolon
- ohitetaan tällä kurssilla.

*Hollantilainen sukunimi, joka kuulemma äännetään "Dykstra".

8.5.1 Suunnatussa syklittömässä verkossa

- Aloitetaan erikoistapauksella: lyhyimmillä poluilla *painotetun suunnatun syklittömän verkon* alkusolmusta $s \in V(G)$.
- Sallitaan myös negatiiviset kaaripainot.

  Negatiivisia(kaan) syklejä ei ole.
- Voidaan käyttää samanlaista menetelmää kuin kalvoilla 8.4.1.5 *painotettujen pisimpien* polkujen etsintään.
- Nyt kenttien arvot kopioituvatkin kaaria pitkin *eteen*päin:
  $$v.\mathrm{dist} = \min\left\{w + u.\mathrm{dist} : u \xrightarrow{w} v \in E(G)\right\}$$
  Eli kenttä $v.\mathrm{dist}$ voidaan laskea vasta kun kaikkien *edeltäjien* kentät $u.\mathrm{dist}$ on laskettu.

1: **for all** $u \in V(G) \setminus \{s\}$ **do**
2: $u.\text{dist} := +\infty$;
3: $u.\pi :=$ NULL
4: **end for;**
5: $s.\text{dist} := 0$;
6: $s.\pi :=$ NULL;
7: Luettele verkon G solmut topologisessa järjestyksessä $p_1, p_2, p_3, \ldots, p_{|V(G)|}$ (esimerkiksi) kalvojen 8.4.1.4 algoritmilla;
8: **for all** $i := 1$ **to** $|V(G)|$ **do**
9: **for all** $p_i \xrightarrow{w} q \in E(G)$ **do**
10: $c := w + p_i.\text{dist}$;
11: **if** $c < q.\text{dist}$ **then**
12: $q.\text{dist} := c$;
13: $q.\pi := p_i$
14: **end if**
15: **end for**
16: **end for.**

- Ajantarve kalvojen 8.3 vieruslistoilla tuttu

$$\mathcal{O}(|V(G)| + |E(G)|).$$

- Rivien 8–16 silmukalla on invariantti:

  Kun käsitellään solmua p_i
  niin $p_j.\text{dist} = \delta(s, p_j)$ *kaikilla* $1 \le j \le i$.

  - Väite pätee kaikilla $1 \le j < i$

    triviaalisti kun $i = 1$

    induktiivisesti kun $i > 1$.

  - Rivin 7 perusteella kaikki solmuun p_i tulevat kaaret $p_k \xrightarrow{w_k} p_i \in E(G)$ on käsitelty jo aikaisemmin (eli $k < i$).

  - Silloin käytettiin näitä oikeita arvoja $p_k.\text{dist} = \delta(s, p_k)$.

  - Jos $p_i = s$ niin

    * jokainen $\delta(s, p_k) = +\infty$ syklittömyyden

    * ja $p_i.\text{dist} = 0$ rivin 5 alustuksen vuoksi.

  - Jos $p_i \neq s$ niin $p_i.\text{dist}$ on kaikkien $w_k + p_k.\text{dist}$ minimi

    * rivin 2 alustuksen

    * ja rivien 10–14 päivityksen vuoksi.

- Invariantin nojalla algoritmi laskee kentät $p_j.\text{dist}$ oikein.

- Rivillä 13 solmun $q \neq s$ kenttä $q.\pi$ asetetaan (ensimmäiseen) sellaiseen p_i joka asettaa tämän oikean arvon c.

- Silloin polku

$$q \xleftarrow{w_1} q.\pi \xleftarrow{w_2} q.\pi.\pi \xleftarrow{w_3} \cdots \xleftarrow{w_m} s$$

  on jokin lyhyin polku alkusolmusta s solmuun q, koska luvut

$$\begin{aligned} q.\text{dist} &= w_1 + q.\pi.\text{dist} \\ q.\pi.\text{dist} &= w_2 + q.\pi.\pi.\text{dist} \\ &\vdots \\ s.\text{dist} &= 0 \end{aligned}$$

  ovat invariantin mukaan oikein.

8.5.2 Dijkstran algoritmi

- Jos (suunnatun tai suuntaamattoman) verkon G kaaripainot ovat kokonaislukuja $w \ge 1$ niin lyhyimmät polut alkusolmusta s voidaan löytää seuraavasti:

  1. Korvataan jokainen alkuperäinen kaari

$$u \xrightarrow{w} v \in E(G)$$

     kokonaisella polulla

$$u \to \overbrace{\underbrace{\bigcirc \to \bigcirc \to \cdots \to \bigcirc}_{w-1 \text{ uutta } \textit{väli}\text{solmua}} \to}^{w \text{ uutta painotonta kaarta}} v.$$

  2. Sovelletaan kalvojen 8.4.2 leveyssuuntaista läpikäyntiä näin saatuun *lavennettuun* verkkoon G'.

- Tämän "ratkaisun" ongelmia:

  - Suurilla w toivottoman hidas ja muistisyöppö.

  - Entä muunlaiset painot w?

- Tämä "ratkaisu" värjää *lavennetun* verkon G' solmuja $p \in V(G')$ **musta**ksi kenttien $p.\mathsf{depth}$ mukaan kasvavassa järjestyksessä.

  Kalvojen 8.4.2 huomion 6 mukaan.

- Tämä "ratkaisu" värjää *alkuperäisen* verkon G solmuja $q \in V(G) \subseteq V(G')$ **musta**ksi kenttien $q.\mathsf{depth}$ mukaan kasvavassa järjestyksessä

- Tämä "ratkaisu" takaa $q.\mathsf{depth} = \delta(s, q)$ sen perusteella, miten alkuperäinen verkko G lavennettiin verkoksi G'.

- Alkuperäiset solmut q värjätään **musta**ksi samassa järjestyksessä
  - jos seuraavaksi värjätään niistä se, jonka $q.\mathsf{dist} = q.\mathsf{depth}$ on *pienin* mahdollinen
  - ilman tarpeettomia välisolmuja $V(G') \setminus V(G)$
  - myös muunlaisille kaaripainoille ≥ 0.

- Siis korvataan työjono Q kalvojen 5.4.3 *prioriteetti*jonolla H kenttien $q.\mathsf{dist}$ suhteen *pienimmästä suurimpaan* päin.

- Myös värierottelu käy tarpeettomaksi:

  Harmaan ja valkean ero ilmoitti aikaisemmin, oliko kentän $p.\mathsf{depth}$ nykyinen arvo jo äärellinen vai vielä alustettu $+\infty$.
  - Nyt myös valkeat solmut voivat aloittaa harmaina
  - koska harmaista solmuista valitaan aina pienin kentän $q.\mathsf{dist}$ suhteen.

  Mustan ja valkean ero ilmoitti aikaisemmin, onko solmu jo käynyt työjonossa Q vaiko ei.
  - Nyt *kaikki* solmut voivat aloittaa prioriteettijonossa H
  - jolloin solmu on joko parhaillaan prioriteettijonossa H tai sitten **musta**.

procedure Dijkstra(G: **verkko**, s: **solmu**)

```
 1: for all u ∈ V(G) \ {s} do
 2:    u.dist := +∞;
 3:    u.π := NULL
 4: end for;
 5: s.dist := 0;
 6: s.π := NULL;
 7: H := buildHeap(V(G));
 8: while not isEmptyHeap(H) do
 9:    u := heapExtractMinimum(H);
10:    for all u -w-> v ∈ E(G) do
11:       c := w + u.dist;
12:       if c < v.dist then
13:          v.dist := c;
14:          v.π := u;
15:          shiftup(H, v)
16:       end if
17:    end for
18: end while.
```

- Muuten kuten kalvojen 8.4.2 leveyssuuntainen läpikäynti ilman värejä

- mutta kenttien päivitys riveillä 13–17 kalvojen 8.5.1 algoritmista

- missä lisärivi 15 käyttää kahvaa alkioon v.

8.5.2.1 Dijkstran algoritmin ominaisuuksia

- Ajantarve:
  - Rivillä 7 luodaan keko H kaikista solmuista $V(G)$.

    Se voidaan tehdä $\mathcal{O}(|V(G)|)$ askeleessa.
  - Rivin 8 keko-operaatio voidaan tehdä $\mathcal{O}(1)$ askeleessa.
  - Rivien 9 ja 15 keko-operaatiot voidaan tehdä $\mathcal{O}(\log |V(G)|)$ askeleessa.
  - Kuen keko-operaatioiden aiheuttama lisäkuormitus yhdistetään kalvojen 8.4.2 huomioon 1 niin saadaan
    - kokonaisajantarpeeksi

      $$\mathcal{O}((|V(G)| + |E(G)|) \cdot \log |V(G)|)$$

      askelta
    - kalvojen 8.3 vieruslistaesityksellä.

Huomioita Dijkstran algoritmista:

1. Kalvojen 8.4.2 huomioon 3 voidaan lisätä kaaripainot:

   Jos $u \xrightarrow{w} v \in E(G)$ niin $\delta(s,v) \leq \delta(s,u) + w$.

2. Kun verkossa G ei sallita negatiivisia syklejä, sen lyhyimmät polut voidaan valita syklittömiksi.

   Todistus. Silloin solmun toistaminen polulla ei lyhennä polun pituutta. □

3. Kaikilla $u \in V(G)$ pätee $u.\text{dist} \geq \delta(s,u)$ rivien 8–18 silmukan ajan.

   - Rivien 1–7 alustukset takaavat sen aluksi. Alkusolmun s alustus rivillä 5 voidaan perustella huomiolla 2.
   - Huomion 1 nojalla rivien 11-16 päivityksetkin säilyttävät sen.

   Voidaan osoittaa kuten kalvojen 8.4.2 huomio 4.

4. Solmun u kenttä on oikein eli $u.\text{dist} = \delta(s,u)$ sillä hetkellä kun solmu u poistuu keosta H rivillä 9.

   Todistus. Tehdään *vastaoletus:* Poistuvalla u kenttä onkin väärin $u.\text{dist} \neq \delta(s,u)$.

   - Voidaan valita *ensimmäinen* virheellinen u.
   - Koska alkusolmu s alustettiin oikein, on $u \neq s$.
   - Huomion 3 nojalla virhe on $u.\text{dist} > \delta(s,u)$.
   - Siis $\delta(s,u) < +\infty$ joten solmu u on saavutettavissa solmusta s.
   - Tarkastellaan oikeaa polkua

     $$\overbrace{\underbrace{s \xrightarrow{w_1} \cdots \xrightarrow{w_k} x}_{\text{alkuosa } p_1} \xrightarrow{w_{k+1}} y \xrightarrow{w_{k+2}} \cdots \xrightarrow{w_\ell} u}^{\text{loppuosa } p_2}$$
     $$\text{kokonaispituus} = \delta(s,u)$$

     missä solmu y on ensimmäinen joka on vielä keossa H.

     Sellainen y on olemassa: viimeistään $y = u$.

   - Koska solmu x on poistunut keosta H jo ennen ensimmäistä virhettä u, niin kenttä $x.\text{dist} = \delta(s,x)$ on oikein.
   - Poistuessaan keosta H solmu x asetti myös kentän $y.\text{dist} = w_{k+1} + x.\text{dist} = \delta(s,y)$ oikein.
   - Siis $u \neq y$, koska u on väärin mutta y oikein.
   - Siten täytyy olla $\delta(s,y) \geq u.\text{dist}$:

     silloinhan keosta H olisi poistettu solmu y eikä solmua u.
   - Miten *koko* oikean polun pituus voisi olla $< u.\text{dist}$, kun jo solmuun y mennessä pituutta on kertynyt yli sen?
   - Vain siten, jos loppuosan p_2 pituus olisi *negatiivinen*.

     Mutta negatiiviset kaarethan ovat kiellettyjä Dijkstran algoritmissa. □

Lause 8.5.1. *Dijkstran algoritmi laskee oikein alkusolmusta s lähtevät lyhyimmät polut*

$$\underbrace{u \xleftarrow{w_1} u.\pi \xleftarrow{w_2} u.\pi.\pi \xleftarrow{w_3} \cdots \xleftarrow{w_m} s}_{\text{kokonaispituus } w_1 + w_2 + w_3 + \cdots + w_m = u.\text{dist}}$$

(suunnatussa tai suuntaamattomassa) verkossa G, jonka kaikki kaaripainot ovat $w_i \geq 0$.

Todistus. Samaan tapaan kuin lause 8.4.3 käyttäen huomiota 4 väitteen 2 roolissa. □

5. Jos Dijkstran algoritmi poistaa solmun p ennen solmua q keosta H, niin $\delta(s,p) \leq \delta(s,q)$.

   - *Todistus.* Suoraan lauseesta 8.5.1 ja keon H vertailuehdosta "$p.\text{dist}$ vs. $q.\text{dist}$". □
   - Joskus halutaan löytää vain alkusolmua s (jokin) *lähin* solmu u (ja polku siihen) joka täyttää annetun ehdon $\phi(u)$.
   - Riville 8 voidaan lisätä ehto **... and** ϕ(heapMinimum(H)) joka lopettaa silmukan *heti* kun sellainen ilmestyy (epätyhjän) keon H päällimmäiseksi.

6. Dijkstran algoritmi voidaan pysäyttää heti kun kaikki alkusolmusta s saavutettavissa olevat solmut u on käsitelty.

   - Valitaan huomiossa 5 ehdoksi $\phi(u)$ testi

     $$u.\mathrm{dist} < +\infty.$$

   - Tämä testi voidaan kirjoittaa myös muodossa

     $$u.\pi \neq \mathrm{NULL} \textbf{ or } u = s.$$

     Enää ei tarvita erillistä vakioarvoa "$+\infty$".

   - Tämä testin muoto käy myös kalvojen 8.4.2 leveyssuuntaiseen läpikäyntiin.

7. Algoritmista saadaan sellainen versio, jonka

   pääsilmukka riveillä 6–20 käyttää aikaa ja tilaa vain suhteessa syöteverkon G *saavutettavan osan* solmujen ja kaarten lukumäärään huomion 6 avulla

   alustussilmukka riveillä 1–3 käy yhä läpi kaikki verkon G solmut $V(G)$.

function finite(u: $V(G)$): **Boolean**
return ($u.\pi \neq$ NULL **or** $u = s$).

```
1:  for all u ∈ V(G) do
2:      u.π := NULL
3:  end for;
4:  s.dist := 0;
5:  H := buildHeap({s});
6:  while (not isEmptyHeap(H)) and
    finite(heapMinimum(H)) do
7:      u := heapExtractMinimum(H);
8:      for all u --w--> v ∈ E(G) do
9:          c := w + u.dist;
10:         if not finite(v) then
11:             v.dist := c;
12:             v.π := u;
13:             heapInsert(H, v)
14:         else if c < v.dist then
15:             v.dist := c;
16:             v.π := u;
17:             shiftup(H, v)
18:         end if
19:     end for
20: end while.
```


8. Huomion 7 alustussilmukastakin voidaan päästä eroon:

   - Otetaan käyttöön aputietorakenne R kaikille alkusolmusta s tähän mennessä saavutetuille solmuille v.
   - Jokaiselle solmulle v talletetaan
     - $v.\mathrm{dist}$ (joka nyt on $< +\infty$)
     - $v.\pi$ (joka nyt on NULL ainoastaan kun $v = s$)
     - kahva solmua v vastaavaan alkioon keossa H riviä 17 varten.
   - Rivillä 5 alustetaan tietorakenne R samoin kuin keko H.
   - Rivin 10 testiksi tulee "solmua v ei vielä löydy tietorakenteesta R".
   - Rivillä 13 lisätään tämä v tietorakenteeseen R.
   - Jos R toteutetaan kalvojen 6.2 tasapainotetulla hakupuulla, niin nopeus pysyy keon H tasolla.

9. Myös kekoa H voidaan vielä tehostaa:

   - Kekoon H tehdään
     - $\mathcal{O}(|V(G)|)$ alkion **poisto**a rivillä 9
     - $\mathcal{O}(|E(G)|)$ avaimen **nosto**a rivillä 15

     ja yleensä $|E(G)| \gg |V(G)|$.
   - Algoritmia voidaan tehostaa (esimerkiksi) *Fibonacci*n keolla H jossa yksi

     poisto vie yhä $O(\log|V(G)|)$

     nosto viekin vain $O(1)$

     askelta *tasoitettuna yli koko suoritusajan*.
     - Siis laskettuna samaan tapaan kuin kalvoilla 7.4.1.
     - Kaikki *nostot kerätään yhteen ja tehdään samalla kerralla* juuri ennen seuraavaa poistoa.
   - Aikavaatimukseksi jää

     $$\mathcal{O}(|E(G)| + |V(G)| \cdot \log|V(G)|)$$

     askelta.

- Peruskeko
  - riittää yleensä käytännössä
  - löytyy useammista tietorakennekirjastoista
  - on helpompi ohjelmoida itse

  kuin Fibonaccin keko
- Jos täytyy *yhdistää* kaksi erillistä vanhaa kekoa toisiinsa yhdeksi uudeksi keoksi, niin
  - Fibonaccin keko toimii *vakio*ajassa (jos vanhoja kekoja ei tarvitse enää säilyttää)
  - peruskeko tarvitsee

    kopiointia jos toteutettu taulukkona kalvojen 5.4.1

    tasapainotuksia jos puuna kalvojen 6.3.4

    tapaan.

8.5.2.2 Epänumeerinen Dijkstra

- Dijkstran algoritmi ei vaadi numeerisia kaaripainoja.
- Samaa perusideaa voidaan soveltaa myös muihin sellaisiin ongelmiin, joissa on kyse "hyvän" polun löytämisestä.
- Määritellään ongelman mukainen vertailuehto:

  "Tämä polku

  $$\mathcal{A} = s \to p_1 \to p_2 \to p_3 \to \cdots \to p_m$$

  on ainakin yhtä hyvä kuin tuo polku

  $$\mathcal{B} = s \to q_1 \to q_2 \to q_3 \to \cdots \to q_n."$$

  Merkitään "ayh$(\mathcal{A}, \mathcal{B})$".
- Tämän määritelmän on täytettävä lisäehto

  ayh$(s \to p_1 \to p_2 \to p_3 \to \cdots \to p_m,$
  $s \to p_1 \to p_2 \to p_3 \to \cdots \to p_m \to p_{m+1})$

  joka vastaa negatiivisten kaarten $p_m \to p_{m+1}$ kieltämistä.

- Vertailuehdon ayh$(\mathcal{A}, \mathcal{B})$ on oltava (kaikkialla määritelty eli) *totaali järjestys* kuten '$\leq$':

  Totaalinen eli aina täytyy olla totta ainakin toinen ehdoista ayh$(\mathcal{A}, \mathcal{B})$ ja ayh$(\mathcal{B}, \mathcal{A})$.

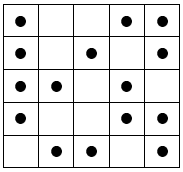
  Antisymmetrinen eli jos molemmat niin polkujen $\mathcal{A}$ ja $\mathcal{B}$ välillä ei tehdä eroa.

  Transitiivinen eli jos ayh$(\mathcal{A}, \mathcal{B})$ ja ayh$(\mathcal{B}, \mathcal{C})$ niin myös ayh$(\mathcal{A}, \mathcal{C})$.

  Refleksiivinen eli ayh$(\mathcal{A}, \mathcal{A})$
- Käytetään numeerisen vertailun $x < y$ sijasta
  - ehtoa "ayh(x, y) mutta ei ayh(y, x)"
  - keossa H ja päivitystestissä $c < v.\,\mathrm{dist}$
  - polkuversioinaan.
- Lauseen 8.5.1 tapaan voidaan päätellä, että näin löydetään jokaiselle solmulle $u \in V(G)$ ainakin yhtä hyvä polku kuin muut tarjolla olleet vaihtoehdot.

8.5.2.3 Dijkstran algoritmi ongelmanratkaisussa

- Monet ongelmat voidaan muuntaa hyvien polkujen löytämiseksi verkosta.
- Tarkastellaan seuraavaa vankilapako-ongelmaa:
  - Syötteenä saadaan vankilan pohjapiirros matriisina $B[0 \ldots 2 \cdot m][0 \ldots 2 \cdot m]$.
  - Jokainen paikka $B[i][j]$ on joko *käytävä*ä tai *muuri*a.



- Olemme aluksi vankilan keskipaikassa $B[m][m]$ (joka on käytävää).
- Tavoitteena on päästä vankilasta sen jonkin reunan yli vapauteen.

- Voimme liikkua vankilassa seuraavasti:
  - Nykyisestä paikasta voi siirtyä mihin tahansa naapuripaikkaan, ei kulmittain.
  - Käytävillä voi hiiviskellä, mutta niillä *ei kannata maleksia* turhaan.
  - Muuriin täytyy ensin *kaivautua* ennen kuin siihen voi kulkea.
- Hyvä pakosuunnitelma on sellainen, jossa on mahdollisimman vähän
  1. kaivamista
  2. käytävillä hiiviskelyä

  tässä järjestyksessä.

  Siis pakosuunnitelma $\mathcal{A}$ on aidosti parempi kuin $\mathcal{B}$ jos
  1. joko $\mathcal{A}$ sisältää vähemmän kaivamista kuin $\mathcal{B}$
  2. tai ne sisältävät yhtä paljon kaivamista, mutta $\mathcal{A}$ sisältää vähemmän käytävillä hiiviskelyä kuin $\mathcal{B}$.

- Mallinnetaan paon suunnittelu lyhyimpänä polkuna verkossa G:
  - $V(G) =$ vankilan paikat sekä lisäpaikka t eli "vapaudessa".
  - $p \xrightarrow{w} q \in E(G)$ jos ja vain jos paikka q on paikan p naapuripaikka.

    Jokaisen reunapaikan p naapurina on myös lisäpaikka t.
  - Kaaripainon $w \geq 0$ määrää kohdepaikka q:

    käytävällä $w = 1$

    muurilla $w = (2 \cdot m + 1)^2$ eli vankilan koko pinta-ala

    reunalla $w = 0$ kun $q = t$.
  - Kalvojen 8.5.2.1 kehättömyyshuomio 2 oikeuttaa nämä kaaripainojen valinnat:
    * Yksikin muuriin kaivautuminen on huonompi kuin pisinkään mahdollinen käytävillä hiiviskely.
    * Samaa muuriin kaivettua aukkoa ei kuljeta kahdesti.

- (Jokin) mahdollisimman hyvä pakosuunnitelma löytyy siis
  - ajamalla Dijkstran algoritmia
  - verkossa G
  - alkusolmusta $s = B[m][m]$ lähtien
  - kunnes keon H päälle tulee t
  - jolloin pakosuunnitelma on polku

$$t \leftarrow t.\pi \leftarrow t.\pi.\pi \leftarrow \cdots \leftarrow s.$$

- Verkkoa G ei tarvitse tallettaa erikseen:
  - Solmun $B[i][j]$ mahdolliset seuraajasolmut ovat $B[i \pm 1, j]$ ja $B[i, j \pm 1]$.
  - Indeksöinnin ylitys tarkoittaa lisäsolmua t.
  - Kaaripaino katsotaan kohdesolmusta.

  Siis G voidaan *laskea* algoritmin edetessä.

- Kalvot 8.5.2.2 mahdollistavat suoremmankin esitystavan:
  - Käytetäänkin luku*pareja* $\langle a, b \rangle$ missä
    * $a =$ muuri-
    * $b =$ käytävä-

    paikkojen lukumäärä polulla.
  - Kaaripainot w pareina

    käytävällä $w = \langle 0, 1 \rangle$

    muurilla $w = \langle 1, 0 \rangle$

    reunalla $w = \langle 0, 0 \rangle$.
  - Pari $\langle a, b \rangle$ on ainakin yhtä hyvä kuin $\langle c, d \rangle$ täsmälleen kun

$$a < c \text{ **or** } (a = c \text{ **and** } b \leq d).$$

  - Polun pituus

$$s \xrightarrow{\langle k_1, m_1 \rangle} p_1 \xrightarrow{\langle k_2, m_2 \rangle} p_2 \xrightarrow{\langle k_3, m_3 \rangle} \cdots$$

    on

$$\langle k_1 + k_2 + k_3 + \cdots, m_1 + m_2 + m_3 + \cdots \rangle .$$

8.5.3 Floydin algoritmi

- Olkoon syöteverkko G talletettu kalvojen 8.3 vierusmatriisina $B[1\ldots|V(G)|][1\ldots|V(G)|]$ seuraavasti:
  - Olemassaoleva kaari $i \xrightarrow{w} j \in E(G)$ esitetään kaaripainona $B[i][j] = w$.
  - Puuttuva kaari esitetään kaaripainona $B[i][j] = +\infty$.
  - Yksittäiset kaaripainot w ovat positiivisia ja negatiivisia lukuja.
  - Negatiivisia syklejä ei sallita.
- Tavoitteena on laskea "välimatkataulukko" $D[1\ldots|V(G)|][1\ldots|V(G)|]$ jossa paikassa $D[i][j]$ on
  - lyhyimmän polun pituus solmusta i solmuun j
  - $+\infty$ jos yhtään sellaista polkua ei ole.

- Käytetään aputietorakenteena 3-ulotteista matriisia eli "kuutiota" $C[1\ldots|V(G)|][1\ldots|V(G)|][0\ldots|V(G)|]$ jonka sisältö määritellään seuraavasti:

  $C[i][j][k]$ = lyhyimmän polun pituus solmusta i solmuun j, kun saa käyttää vain

  solmuja $\mathcal{V}_k = \{i, j\} \cup \{1, 2, 3, \ldots, k\}$

  kaaria näiden solmujen välillä.

  Eli siinä verkon G *aliverkossa* (subgraph) $G_k \subseteq G$ (huomaa merkintäsopimus), jonka *virittää* (induces) solmujen osajoukko $\mathcal{V}_k \subseteq E(G)$.

- Haluttu välimatkataulukko on silloin kuution C *viimeinen* kerros $k = |V(G)|$:

  $$D[i][j] = C[i][j][k].$$

- Kuution C *ensimmäinen* kerros $k = 0$ saadaan vierusmatriisista B:

  diagonaalilla $D[h][h][0] = 0$ kalvojen 8.5.2.1 havainnnolla 2

  muualla $D[i][j][0] = B[i][j]$ missä $i \neq j$.

- Kuution C ylempi kerros $k > 0$ saadaan laskettua edellisestä kerroksesta $k - 1$:
  - Tarkastellaan lyhyintä polkua

    $$\mathcal{P} = i \to \underbrace{\bigcirc \to \bigcirc \to \bigcirc \cdots \bigcirc}_{\text{polun sisäsolmut}} \to j$$

    aliverkossa G_k.
  - Jos sellaista polkua ei ole verkossa G_k, niin sellaista polkua ei ole myöskään sen aliverkossa $G_{k-1} \subseteq G_k$.

    Silloin $C[i][j][k] = C[i][j][k-1] = +\infty$.
  - Jos solmu k ei esiinny polulla sisäsolmuna, niin polku on jo aliverkossa $G_{k-1} \subseteq G_k$.

    Silloin $C[i][j][k] = C[i][j][k-1] < +\infty$.
  - Jos solmu k esiintyy polulla sisäsolmuna, niin tasan *kerran* kalvojen 8.5.2.1 havainnolla 2:

    $$\overbrace{i \to \bigcirc \to \cdots \to \bigcirc \to k}^{\text{polun alkuosa}} \underbrace{\to \bigcirc \to \cdots \to \bigcirc \to j}_{\text{polun loppuosa}}$$

- Näin jaetun polun

  alkuosa $i \to \bigcirc \to \cdots \to \bigcirc \to k$

  loppuosa $k \to \bigcirc \to \cdots \to \bigcirc \to j$
  - kulkevat aliverkossa $G_{k-1} \subseteq G_k$

    koska solmu k ei esiinny niissä sisäsolmuna
  - ovat lyhyimpiä polkuja aliverkossa $G_{k-1} \subseteq G_k$

    koska vielä lyhyemmällä osapolulla voisi lyhentää verkon G_k lyhyimmäksi oletettua polkua $\mathcal{P}$.
- Jaetulle polulle saadaan siis

  $$C[i][j][k] = C[i][k][k-1] + C[k][j][k-1].$$
- Yhdistämällä nämä vaihtoehdot saadaan laskusääntö

  $$C[i][j][k] = \min(C[i][j][k-1], C[i][k][k-1] + C[k][j][k-1]). \quad (32)$$

- Laskusääntö (32) tarvitsee seuraavaan kerrokseen $k > 0$ vain edellisen kerroksen $k - 1$.
  - Sitä edelliset kerrokset $k-2, k-3, k-4, \ldots, 0$ ovat tulleet tarpeettomiksi.
  - Siis voidaan tallettaa taso k tason $k-2$ tilalle.
  - Siis riittää *2-tasoinen* kuutio $C'[1 \ldots |V(G)|][1 \ldots |V(G)|][0 \ldots 1]$ jonka sisältö on
    $$C'[i][j][k \bmod 2] = C[i][j][k].$$
- Näin tilantarve putoaa kuutiosta
  $$\mathcal{O}(|V(G)|^3)$$
  neliöön
  $$\mathcal{O}(|V(G)|^2)$$
  mikä on merkittävä säästö!

- Riittääkö vain *1-tasoinen* kuutio C'' eli pelkkä matriisi?
  - Eli lasketaankin *paikallaan* säännöllä
    $$C''[i][j] = \min(C''[i][j], C''[i][k] + C''[k][j]) \quad (33)$$
    kun $k > 0$.
  - Jos $C''[i][k]$ on jo laskettu tällä samalla k, niin tämä kaava ei enää viittaakaan *edelliseen* arvoon $C[i][k][k-1]$ vaan *nykyiseen* $C[i][k][k]$.
  - Mutta laskusäännön (32) perusteluista yllä seuraa, että
    $$C[i][k][k] = C[i][k][k-1] \quad (34)$$
    joten uusikin laskusääntö (33) on oikein.
  - Samoin $C[k][j]$.

  Siis riittää!
- Silloin voidaan matriisina C'' käyttää suoraan vastaukselle varattua tilaa D.


```
1:  for all i := 1 to |V(G)| do
2:     for all j := 1 to |V(G)| do
3:        if i = j then
4:           D[i][j] := 0
5:        else
6:           D[i][j] := B[i][j]
7:        end if
8:     end for
9:  end for;
10: for all k := 1 to |V(G)| do
11:    for all i := 1 to |V(G)| do
12:       for all j := 1 to |V(G)| do
13:          D[i][j] := min(D[i][j], D[i][k] +
             D[k][j]) laskusäännön (33) mukaan
14:       end for
15:    end for
16: end for.
```

- Aikavaatimus
  $$\Theta(|V(G)|^3)$$
  askelta suoraan sisäkkäisistä silmukoista.

- Laskusäännön (32) perusteluista seuraa että $D[i][j]$ = lyhyimmän *yksinkertaisen* polun pituus solmusta i solmuun j.
- Jos on jokin negatiivinen polku jostakin solmusta takaisin itseensä, niin on myös jokin yksinkertainen sykli:
  - Tarkastellaan negatiivista polkua
    $$i \to \cdots \to \underbrace{j \to \cdots \to j}_{\text{väliosa}} \to \cdots \to i \quad (35)$$
    joka ei ole yksinkertainen.
  - Jos väliosan pituus ≥ 0 niin poistetaan se.
  - Jos väliosan pituus < 0 niin keskitytään siihen.
  - Lopulta jää yksinkertainen sykli.
- Siis Floydin algoritmi voi *tunnistaa* virheellisen syötematriisin B:

  Tulosmatriisin diagonaalille ilmestyy negatiivinen luku $D[j][j] < 0$.