- Entä milloin ylipitkää hajautustaulua T kannattaa *lyhentää* poistojen jälkeen?

  (Jos kertaalleen varattua mutta tarpeettomaksi käynyttä muistia on tarpeen kierrättää muuhun hyötykäyttöön luovuttamalla se takaisin käyttöjärjestelmän/virtuaalikoneen muistinhallinnalle.)

- Vakiolla lyhennys kärsii samasta ongelmasta kuin vakiolla pidennys.

- Käytetään siis edellistä kaksinkertaistusideaa ja *puolitetaan* ylipitkä taulu.

- Jos puolitetaan heti kun kun $\alpha = \frac{1}{2}$, niin

  1. Puolityhjän taulun puolitus tuottaa täyden taulun.
  2. Siihen avaimen lisääminen aiheuttaa pidennyksen.
  3. Siitä avaimen poisto aiheuttaa puolituksen ja täyden taulun.
  4. Siihen avaimen lisääminen...

- Samanlainen vakiomäärä lyhennystyötä operaatiota kohden saadaan puolittamalla *vasta* kun

$$\alpha = \frac{1}{4}.$$

- Saadaan periaate:

  Kaksinkertaista taulukko kun siihen ei enää mahdu seuraavaa lisättävää avainta.

  Puolita taulukko kun siitä on käytössä enää $\frac{1}{4}$.

  Silloin

  – keskimääräinen lisätyö operaatiota kohden on vakio

  – vaikka silloin tällöin jokin operaatio onkin hidas.

- Samaa periaatetta voi soveltaa myös kalvojen 5.4.1.2 taulukkoon talletulle keolle.

  Niin saadaan taulukkopohjainen vaihtoehto kalvojen 6.3.4 puupohjaiselle ylärajattomalle prioriteettijonolle.

7.4.2 Kokeilujonoja

Lineaarinen kokeilu (linear probing)

- käyttää tavallista hajautusfunktiona

$$h' \colon U \to \{0, 1, 2, \ldots, m-1\}$$

  *apu*funktiona (auxiliary function), joka määrää jonon alkukohdan

- kokeilee alkukohdasta alkaen paikkoja järjestyksessä

$$T[h'(x)], T[h'(x)+1], T[h'(x)+2], \ldots$$

- pyörähtää taulukon T alkuun ohitettuaan sen lopun

eli on muotoa

$$h(x,i) = (h'(x) + i) \bmod m.$$

Neliöillinen kokeilu (quadratic probing)
laajentaa lineaarisen kokeilun *toisen asteen* polynomiksi

$$h(x,i) = (h'(x) + c_1 \cdot i + c_2 \cdot i^2) \bmod m$$

sopivilla vakioilla c_1 ja $c_2 \neq 0$.

- Yksi tapa on seuraava:

  1. Aluksi asetetaan $p := h(x)$; $q := 0$.
  2. Tutkitaan paikka $T[p]$.
  3. Jos täytyy jatkaa, niin asetetaan $q := (q+1) \bmod m$; $p := (p+q) \bmod m$ ja palataan askeleeseen 2.

  Hajautustaulun pituus on muotoa $m = 2^b$.

- Toinen tapa on

$$h(x,i) = \left(h'(x) + (-1)^i \cdot \left\lfloor \frac{i+1}{2} \right\rfloor^2\right) \bmod m$$

Kaksoishajautus (double hashing) käyttää apufunktioina *kahta* tavallista hajautusfunktiota h_1 ja h_2:

$$h(x,i) = (h_1(x) + i \cdot h_2(x)) \bmod m.$$

- Parempi kuin lineaarinen tai neliöllinen kokeilu:

  Kokeilujono riippuu *kahdella* eri tavalla avaimesta x.

- Arvoilla m ja $h_2(x)$ ei saa olla yhteisiä tekijöitä:

  muuten osa taulusta T jää käymättä läpi.

  - Taulun pituus voi olla muotoa $m = 2^b$, ja $h_2(x)$ pariton.

  - Tai m voi olla alkuluku, ja $h_2(x) < m$.

    Esimerkiksi

    $$h_1(x) = k \bmod m$$
    $$h_2(x) = 1 + (x \bmod m')$$

    missä valitaan jokin (suuri) $m' < m$ (vaikkapa $m' = m - 1$).

7.4.3 Avoimen hajautuksen analyysistä

- Oletetaan (kalvojen 7.1 tapaan) yksinkertaisuuden vuoksi *yhdenmukainen kokeilu* (uniform hashing):

  - Avaimen x kokeilujono valitaan tarjolla olevista $m!$ permutaatiosta siten, että jokainen niistä on yhtä todennäköinen.

  - Kokeilumenetelmänä

    teoriassa optimaalinen

    käytännössä liian vaikea.

    Miten lähelle kalvojen 7.4.2 menetelmät pääsevät?

  - Tutkittujen taulukkopaikkojen lukumäärän odotusarvo on

    tuloksettomassa haussa $\leq \frac{1}{1-\alpha}$

    tuloksellisessa haussa $\leq \frac{1}{\alpha} \cdot \ln \frac{1}{1-\alpha}$

    missä $\alpha < 1$.

    (Todennäköisyyslaskelmat sivuutetaan.)

Lineaarisella kokeilulla nämä odotusarvot ovat

tuloksettomassa $\frac{1}{2} \cdot \left(1 + \frac{1}{(1-\alpha)^2}\right)$

tuloksellisessa $\frac{1}{2} \cdot \left(1 + \frac{1}{1-\alpha}\right)$.

Syynä on *ensisijainen kasautuminen* (primary clustering):

- Paikkaan $T[p]$ törmäävät q avainta varaavat peräkkäiset paikat

  $$T[p], T[p+1], T[p+2], \ldots, T[p+q-1]$$

- Tämä pakottaa näihin varattuihin paikkoihin kuuluvat avaimet varatun alueen loppuun

  $$T[p+q], T[p+q+1], T[p+q+2], \ldots$$

- Siis yhtenäinen varattu alue kasvaa vielä pidemmäksi sitä todennäköisemmin mitä pidempi se jo nyt on!

Neliöllisellä kokeilulla nämä odotusarvot ovat

tuloksettomassa $\frac{1}{1-\alpha} - \alpha + \ln \frac{1}{1-\alpha}$

tuloksellisessa $1 - \frac{\alpha}{2} + \ln \frac{1}{1-\alpha}$

Syynä on *toissijainen kasautuminen* (secondary clustering):

- Jos $h'(x) = h'(y)$ niin yhteen törmänneiden avainten x ja y kokeilujono on sama.

- Menetelmä siis

  - jakaa ylivuotoketjut ympäri hajautustaulua paremmin kuin lineaarinen kokeilu

  - mutta pitää ketjut edelleen koossa.

Kaksoishajautuksella nämä odotusarvot ovat samanlaiset kuin yhdenmukaisella kokeilulla.

Toinen hajautus tallettaa avaimen x ylivuotoketjun eri tavoin kuin avaimen y.

8 Verkot

- *Verkko* (graph) G koostuu 2 joukosta:

  Solmujen (node, vertex (monikko "vertices")) joukosta $V(G)$.

  Oletamme solmut numeroiduiksi $1, 2, 3, \ldots, |V(G)|$.

  Kaarten (edge, arc) joukosta $E(G) \subseteq V(G) \times V(G)$.

  – Yksi kaari $\langle p, q \rangle \in E(G)$ on siis *solmupari*.

  – Merkitään kaarta $p \to q$.

- Kalvojen 5.1.3.1 binääri- ja yleisemmät puut ovat verkkojen erikoistapauksia.

- Verkkoja käytetään (puiden tapaan)

  ongelman mallintamisen abstrakteina

  informaation tallentamisen konkreettisina

  välineinä.

Kaaren $p \to q$ solmut ovat *vierekkäisiä* (adjacent) eli *vierussolmuja*.

- Solmu p *edeltää* solmua q.
- Solmu q *seuraa* solmua p.

Solmun lähtöaste (outdegree, out-degree) on *siitä lähtevien* kaarten lukumäärä.

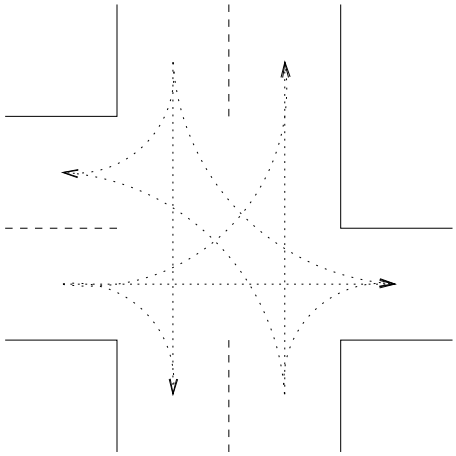
Tuloaste (indegree, in-degree) on *siihen tulevien* kaarten lukumäärä.

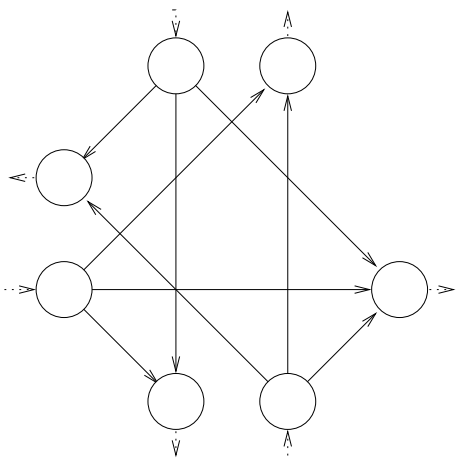
- Tästä perusteemasta on paljon muunnelmia eri käyttötarkoituksiin.

- Esimerkkinä kaupungin *katuverkko:*

  – Kukin kaari esittää sellaista ajokaistan pätkää, joka auton on ajettava alusta loppuun, koska välillä ei voi kääntyä.

  – Kukin solmu esittää sellaista kohtaa, jossa auto voi valita seuraavan suuntansa useasta eri ajokaistan pätkästä.





8.1 Painottamattomat ja painotetut verkot

- Kalvoilla 8 määriteltiin *painottamaton* (unveighted) verkko.

- *Painotetun* (weighted) verkon G jokaisella kaarella on oma kaari*paino* (edge *weight*) w:

$$p \xrightarrow{w} q \in E(G).$$

- Paino w voi olla

  luku — kuten

  – etäisyys

  – siirtokapasiteetti

  – ...

  merkki(jono) — kuten äärellisissä automaateissa

  ...

- Painotetussakin verkossa sallitaan yleensä korkeintaan *yksi* kaari

  lähtösolmusta (source) $p \in V(G)$

  maalisolmuun (sink) $q \in V(G)$.

- *Moni*verkossa (*multi*graph) sallitaan *monta* sellaista kaarta, yksi jokaiselle eri painolle w.

  Esimerkiksi äärellisissä automaateissa.

Polku eli **reitti** on kaarijono

$$p_0 \xrightarrow{w_1} p_1 \xrightarrow{w_2} p_2 \xrightarrow{w_3} \cdots \xrightarrow{w_m} p_m$$

missä jokainen solmu p_{i+1} seuraa solmua p_i.

Solmusta $v \in V(G)$ **saavutettavissa** (reachable, accessible) ovat ne solmut $w \in V(G)$ joihin on polku solmusta v.

Kun painot esittävät kaarten pituuksia niin voidaan puhua solmujen välisistä *etäisyyksistä* polkuja pitkin.

Polun pituus on

$$w_1 + w_2 + w_3 + \cdots + w_m$$

kun kaaripainot w_i ovat numeroita.

Painottamattomassa verkossa polun pituus määritellään

- siinä olevien *kaarten lukumääräksi* m
- ikään kuin kaikki kaaripainot olisivat

$$w_1 = w_2 = w_3 = \cdots = w_m = 1.$$

Yksinkertainen (simple) polku on sellainen, jossa mikään solmu ei toistu.

Sykli eli **kehä** (cycle) on sellainen polku

- joka on muuten yksinkertainen
- mutta sen alku- ja loppusolmu ovat samat $p_1 = p_m$.

DAG (**D**irected **A**cyclic **G**raph) on suosittu lyhenne *suunnatulle syklittömälle verkolle*.

8.2 Suunnatut ja suuntaamattomat verkot

- Kalvoilla 8 määriteltiin *suunnatut* (directed) verkot:

  Verkon G kaarella

  $$p \to q \in E(G)$$

  on *luonteva kulkusuunta* lähtösolmusta p maalisolmuun q.

- *Suuntaamattomassa* (undirected) verkossa kaarella ei ole suuntaa

  $$\{p, q\} \in E(G)$$

  eikä sen päitä p ja q erotella toisistaan.

  Suuntamatonta kaarta aletaan kutsua **särmä**ksi.

- Kalvojen 8 käsitteet soveltuvat suuntaamattomiin verkkoihin.

  Mutta tulo- ja lähtöasteiden sijasta puhutaankin vain **aste**esta (degree).

8.3 Verkkojen talletusrakenteita

Vieruslistat (adjacency lists):

- Taulukko $A[0 \ldots |V(G)| - 1]$ jonka paikka $A[p]$ sisältää solmun $p \in V(G)$
  - vieruslistan
  - solmukohtaiset muut tiedot (jos on).
- Solmun $p \in V(G)$ vieruslista sisältää siitä lähtevät kaaret:
  - Kaarelle $p \xrightarrow{w} q \in E(G)$ parin $\langle w, q \rangle$ (sopivana tietueena).
  - Painottamattomalla G ei kenttää w.
- Hyvä tapa kun:
  - Verkkoa G käsitellään *solmu kerrallaan*.
  - Verkko G on *harva* (sparse), eli sisältää vain "pienen" osan kaikista mahdollisista kaarista, eli suhde

    $$\frac{|E(G)|}{|V(G)|^2}$$ on pieni.

Vierusmatriisi (adjacency matrix):

- 2-ulotteinen matriisi (taulukko taulukoita) $B[1 \ldots |V(G)|][1 \ldots |V(G)|]$.
- Paikassa $B[p][q]$ on solmujen p ja q välinen tieto:

  Painottamattomalla verkolla G *totuusarvo*

  $$p \to q \in E(G)?$$

  Painotetulla verkolla G

  – vastaava kaaripaino w, jos

  $$p \xrightarrow{w} q \in E(G)$$

  – Jokin sopiva "mahdoton paino" $\aleph$, jos

  $$p \xrightarrow{w} q \notin E(G).$$

  Jos esimerkiksi lukuarvoiset painot w esittävät etäisyyksiä, niin $\aleph = +\infty$ on luonteva valinta.

- Vierusmatriisi on hyvä tapa kun:
  – Verkkoa G käsitellään *kaari sieltä, toinen täältä*.
  – Verkko G onkin *tiheä* (dense).

Suuntaamaton verkko G voidaan esittää *molempiin suuntiin suunnattuna*, eli kaariparina

$$p \rightleftarrows q \in E(G)$$

jokaiselle särmälle solmujen p ja q välillä.

Tapoja, jotka eivät 2-kertaista särmiä:

***Kolmio*matriisina** (*triangular* matrix)

$$C[\underbrace{1 \ldots |V(G)|}_{\text{indeksi } i}][1 \ldots i]$$

missä solmujen p ja q välinen särmä on paikassa $C[\max(p,q)][\min(p,q)]$.

***Moni*listoilla** (*multi*lists) joissa solmujen p ja q välisen särmän lista-alkiossa on *molemmat* solmut ja listojen $\max(p,q)$ ja $\min(p,q)$ osoittimet.

8.4 Verkon järjestelmällinen läpikäynti

- Käydään annettu verkko G läpi
  – pelkästään sen solmu- ja kaari*rakenteen* (eli *topologian*) ohjaamana
  – jättäen mahdolliset kaaripainot huomiotta.

- 2 perustapaa:

  syvyyssuuntainen eli "*Rohkeasti* eteenpäin tätä solmua seuraaviin vielä tuntemattomiin solmuihin!"

  leveyssuuntainen eli "*Varovasti* eteenpäin kotisolmua lähimpiin vielä tuntemattomiin solmuhin..."

  Kuten kalvoilla 6.3.4.2 (binääri)puille.

- Verkon G solmun $u \in V(G)$ väri $u.\text{colour}$ on

  WHITE kunnes siihen tullaan ensimmäisen kerran

  BLACK kun kaikki sen seuraajatkin on käsitelty

  GRAY väliajan kun sen seuraajien käsittely on vielä kesken.

  Näillä pidetään huolta, että kukin *solmu käsitellään vain kerran*.

- Läpikäynti luo samalla π*-puun:*
  – Kaari $p \to q$ π-puussa tarkoittaa:
  Läpikäynti löysi solmun $p \in V(G)$ *ensimmäistä* kertaa kaarta $q \to p \in E(G)$ pitkin.
  – Siis π-puussa kaaret menevätkin *lapsesta isään*.

8.4.1 Syvyyssuuntainen läpikäynti

- Verkon G *syvyys*suuntainen läpikäynti (Depth-First Search, DFS) on keskeinen työkalu monissa verkkoalgoritmeissa.

- Läpikäynnin aikana verkon solmuun $u \in V(G)$ talletetaan *"kellonajat":*

  Alkuaika $u.\text{started}$ = milloin u muuttui valkeasta harmaaksi

  Loppuaika $u.\text{stopped}$ = milloin u muuttui harmaasta mustaksi.

  Ne antavat paljon informaatiota verkosta G.

- Tavallisesti syvyyssuuntainen läpikäynti tehdään käyden verkon *jokaisessa* solmussa (tasan kerran).

- Yleensä saadaan *joukko* π-puita eli *metsä* (forest).

- Jokainen π-puu on kalvojen 5.1.3.2 rekursiokutsupuu.

procedure DFS(G: **verkko**)
  for all $u \in V(G)$ **do**
    $u.\text{colour}$:= WHITE;
    $u.\pi$:= NULL
  end for;
  Globaali time := 0;.
  for all $u \in V(G)$ **do**
    if $u.\text{colour}$ = WHITE **then**
      DFSVisit(u)
    end if
  end for.

procedure DFSVisit(u: $V(G)$)

```
 1: u.colour := GRAY;
 2: time := time+1;
 3: u.started := time;
 4: for all u → v ∈ E(G) do
 5:    if v.colour = WHITE then
 6:       v.π := u;
 7:       DFSVisit(v)
 8:    end if
 9: end for;
10: u.colour := BLACK;
11: time := time+1;
12: u.stopped := time.
```


- Verkko G kannattaa tallettaa kalvojen 8.3 vieruslistoina:
  - Rivien 4–9 silmukka käy läpi koko listan $A[u]$.
  - Vierusmatriisilla jouduttaisiin käymään läpi myös tyhjät paikat $B[u][v]$.

- Silloin koko läpikäynti vie

$$\Theta(|E(G)| + |V(G)|)$$

  askelta:
  - Aliohjelmaa kutsutaan *kerran* jokaiselle solmulle $p \in V(G)$:
    1. Kutsuttaessa $p.\text{colour}$ = WHITE.
    2. Kutsun aluksi p tummuu.
    3. Eikä p enää koskaan vaalene.
  - Aliohjelma(n silmukka riveillä 4–9) vie

$$\Theta(\text{solmun } u \text{ lähtöaste})$$

    askelta ilman rekursioissa vietettyä aikaa.