MAT.APP.270 Kevät 2024
Harjoitus 6:

1. Etsi kirjallisuudesta Floyd-Warshall- algoritmi ja esitä sen toimintaperiaate. Käytä tarvittaessa pseudokoodia.

Floyd-Warshall- algoritmilla voidaan löytää joko suunnatusta tai suuntaamattomasta painotetusta graafista lyhyimmät polut kaikkien solmujen väliltä. Algoritmi ei toimi, jos graafissa on negatiivisia silmukoita, mutta se toimii kuitenkin, jos graafissa on negatiivisen painon kaaria. Algoritmi toimii vertailemalla mahdollisia polkuja kaikkien solmuparien välillä. Algoritmilla voidaan luoda etäisyysmatriisi graafin solmuista, joka kertoo etäisyyden solmujen välillä. Käydään algoritmi läpi pseudokoodin avulla:

```
olkoon dist |V| x |V| vähimmäisetäisyyksien joukko, jotka on alustettu
äärettömiksi.

// Alusta etäisyysmatriisin kaarien painoilla
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // Kaaren (u, v) paino

// Aseta pienimmäksi etäisyydeksi solmusta itseensä 0
for each vertex v do
    dist[v][v] ← 0

// Dynamic Programming Iteration
for k from 1 to |V| do // k uloimassa loopissa edustaa välisolmuja poluissa
    for i from 1 to |V| do
        for j from 1 to |V| do
        // Jos etäisyys solmujen välillä on pitempi kuin välisolmun k kautta
        // niin päivitetään uusi lyhyempi etäisyys etäisyysmatriisiin
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
        end for
    end for
end for

// Algoritmin päätteeksi etäisyysmatriisi dist pitää sisällään lyhyimmät
etäisyydet kaikkien solmujen välillä
```

2. Selitä sitten miten Floyd-Warshall algoritmi soveltuu annetun relaation *transitiivisen sulkeuman* etsimiseen.

Floyd-Warshall algoritmi löytää lyhyimmät etäisyydet kaikkien graafin solmujen välillä. Jos kahden solmun välillä ei ole mahdollista polkua niin algoritmin rakentama etäisyysmatriisi kertoo, että solmujen välinen etäisyys on ääretön. Algoritmia voi siis hyödyntää myös löytämään mihin kaikkiin graafin solmuihin kustakin solmusta pääsee, eli etsimään sen transitiivisen sulkeuman.

3. Valitse jokin reaalimaailman graafimalli (sosiaalisista verkoista, liikenteestä, tietoverkoista, tms) ja jokin sentraalisuusmitta. (EI kuitenkaan mikään, mikä on käsitelty prujussa, kuten PageRank tai Betweennes)
Selitä sanallisesti, kytkien varsinaiseen määritelmään, mitä sentraalisuuden mitta kertoo mallinnettavasta ilmiöstä. Anna yksinkertainen esimerkki, jossa on noin 5-10 solmua. Mainitse lähde, jota käytit

Esimerkkinä reaalimaailman graafista meillä on esim. lento- tai junayhteydet asemien välillä ja sentraalisuusmittana harmonic centrality eli harmoninen sentraalisuus. Harmonic centrality on sentraalisuuden mitta, se on variantti closeness centralitystä ja se kehitettiin ratkaisemaan ongelma mikä closeness centralityllä oli ei-kytktetyillä graafeilla. Harmonic centrality lasketaan seuraavasti:

$$H(v) = \sum_{u|u \neq v} \frac{1}{d(u,v)}$$

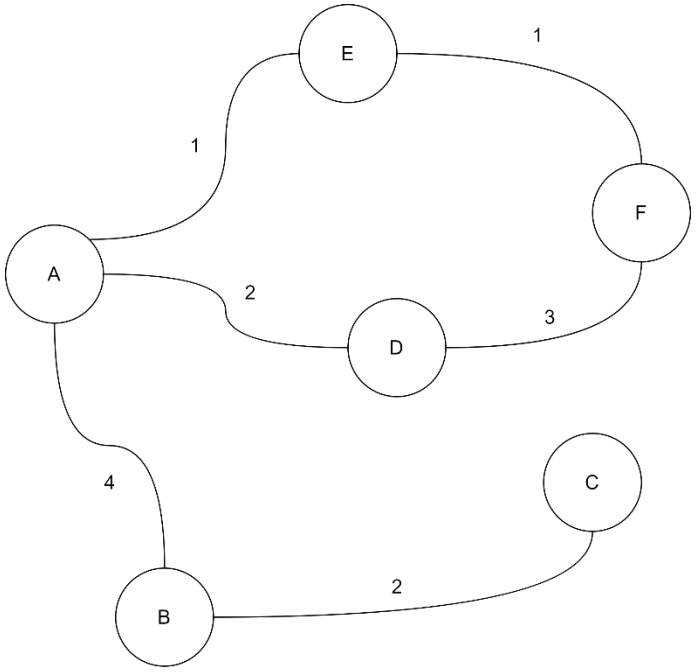
Jos solmujen u ja v välillä ei ole polkua niin 1/d(u, v) = 0. Solmu jolla on suuri harmoninen sentraalisuus on sellainen joka on mahdollisimman lähellä mahdollisimman monta solmua, niiden lyhyimpien etäisyyksien perusteella.

Kytkien sentraalisuusmitta esimerkkitapaukseen asemista, niin asema, jolla on suuri harmonisen sentraalisuuden arvo, on sellainen, joka on mahdollisimman lähellä mahdollisimman monta muuta asemaa. Eli mahdollisimman keskeinen asema olemassa olevassa liikennöinti verkossa.

Lähde:

https://neo4j.com/docs/graph-data-science/current/algorithms/harmonic-centrality/

Esimerkki:



Lyhyimmät etäisyydet solmujen välillä:

	A	B	C	D	E	F	$\sum_y d(y,x)$
A	0	4	6	2	1	2	15
B	4	0	2	6	5	6	23
C	6	2	0	8	7	8	31
D	2	6	8	0	3	3	22
E	1	5	7	3	0	1	17
F	2	6	8	3	1	0	20

Harmoniset sentraalisuudet solmuille:

A: 1/15, **B**: 1/23, **C**: 1/31, **D**: 1/22, **E**: 1/17, **F**: 1/20

Siis solmu A on tämän sentraalisuusmitan mukaan graafin kaikkein sentraalisin solmu.

4. Tutustu prujussa esitettyyn Brandes- algoritmiin. Siinä on vaihe BRANDES-BFS, jonka aikana pinoon S laitetaan alkioita siinä järjestyksessä kun BFS ottaa ne jonosta.

   (a) Osoita, että kun v tulee jonosta BRANDES-BFS:n aikana niin $\sigma(s, v)$ on lyhimpien solmusta s solmuun v johtavien polkujen lukumäärä.

   (b) Miksi $\sigma(s, u) \geq \sigma(s, v)$ jos $v \in \text{PRED}[u]$?

a) BRANDES-BFS algoritmi prujusta:

```
1 queue Q := {s}
2 d[s] := 0
3 σ(s,s) := 1
4 while Q ≠ ∅ do
5     v := Q.dequeue()
6     S.push(v)
7      for w ∈ adj(v)
8            if d[w] = ∞ then
9                 d[w] := d[v] + 1
10                Q.enqueue(w)
11           end if
12          if d[w] = d[v] + 1 then
13                σ(s, w) := σ(s,w) + σ(s,v)
14                Pred[w].push(v)
15          end if
16     end for
17 end while
```

Algoritmissa merkintä d[v] tarkoittaa solmun v etäisyyttä solmusta s. Riveillä 8-9 vieraillaan solmun v naapuri solmuissa w jos niissä ei ole jo vierailtu, päivitetään niiden etäisyys solmusta s ja lisätään ne jonoon vierailua varten. Riveillä 12-13 sen sijaan katsotaan jos solmun w etäisyys solmusta s on yksi enemmän kuin solmun v etäisyys solmusta s, jos tämä pitää paikkansa niin ollaan löydetty yksi uusi lyhyin polku solmuun w solmun v kautta. Siis solmun w lyhyimpiin polkuihin lisätään solmun v lyhyimpien polkujen määrä. Koska käytetään BFS algoritmia niin kun solmu w otetaan pois jonosta solmuna v, on kaikkia sitä edeltävissä solmuissa jo vierailtu eikä ole mahdollista löytää enää lisää lyhyimpiä polkuja solmuun v. Sillä kaikki jäljellä olevat solmut josta voisi päästä solmuun v ovat kauempana solmusta s kuin mitä solmu v on.

b) Lyhyimpien polkujen lukumäärän on oltava yhtä suuri tai suurempi solmulle u kuin solmulle v, koska algoritmissa rivillä 13 solmun u lyhyimpien polkujen lukumäärä on lyhyimpien polkujen lukumäärä solmuun v mihin lisätään muiden solmuun u löydettyjen polkujen lukumäärä. Solmu v kuuluu siis vähintään yhteen lyhyimmistä poluista solmuun u solmusta s, mutta voi myös olla useampia yhtä lyhyitä polkuja jonkun muun solmun kautta, mutta lyhyimpiä polkuja on kuitenkin vähintään yhtä monta kuin niitä on solmuun v.

5. Selvitä miten A^*-algoritmi toimii. Tehtävässä on kaksi osaa

   (a) Kerro, mitä tarkoittaa *heuristiikka* jota A^*-algoritmissa käytetään, ja anna siitä esimerkki reaalimaailman tilanteesta.

   (b) Selitä miten Dijkstran algoritmia tulee muokata jotta saadaan A^*. Koita pohtia myös miten oikeellisuuden todistaminen muuttuu.

a) Heuristiikka tarkoittaa jotain heuristista funktiota, jolla arvioidaan ”hintaa” käsittelyssä olevasta solmusta maali solmuun. A* algoritmissa solmun ”kokonaishinnaksi” f(n) arvioidaan sen varsinainen etäisyys aloitus solmusta käsittelyssä olevaan solmuun n eli g(n) ja sen heuristisen arvion h(n) summaksi, eli solmun kokonaishinta on f(n) = g(n) + h(n).

Esimerkki reaalimaailman tilanteesta voisi olla tilanne, jossa robotti yrittää etsiä lyhyintä mahdollista reittiä varastossa missä on esteitä reitillä. Heuristisena arviona voisi käyttää lyhyintä mahdollista etäisyyttä eli suoraa linjaa / linnuntietä robotin nykyisestä sijainnista maaliin.

b) Jokaiselle solmulle on lisättävä heuristinen arvio h(n) jolla arvioidaan solmun hintaa jollain funktiolla ohjaamaan polun etsimistä. Kuten Dijkstran algoritmissa niin käytetään jonkin laista priority queue -tyyppistä rakennetta päättämään mikä solmu tutkitaan seuraavaksi, mutta kokonaishintana käytetään edellisessä kohdassa määriteltyä kokonaishintaa eli todellisen etäisyyden ja heuristisen arvion summaa. Oikeellisuuden todistaminen A* algoritmissa muuttuu hieman, koska pitää huomioida heuristisen arvion vaikutus siihen, miten algoritmi päättää missä järjestyksessä solmuja tutkitaan. A* algoritmi on käytännössä Best First Search -algoritmi ja oikeellisuuden todistaminen perustuu siihen että voidaan osoittaa että A* tosiaan tutkii solmuja sellaisessa järjestyksessä mikä minimoi polun estimoidun kokonaishinnan.

6. Olkoon (V, E) (suuntaamaton) kytketty graafi. Sanomme että kaari $(u, v) \in E$ on *silta* jos $(V, E \setminus \{(u, v)\})$ ei ole kytketty, eli kaaren poistaminen tekee graafista ei-kytketyn.

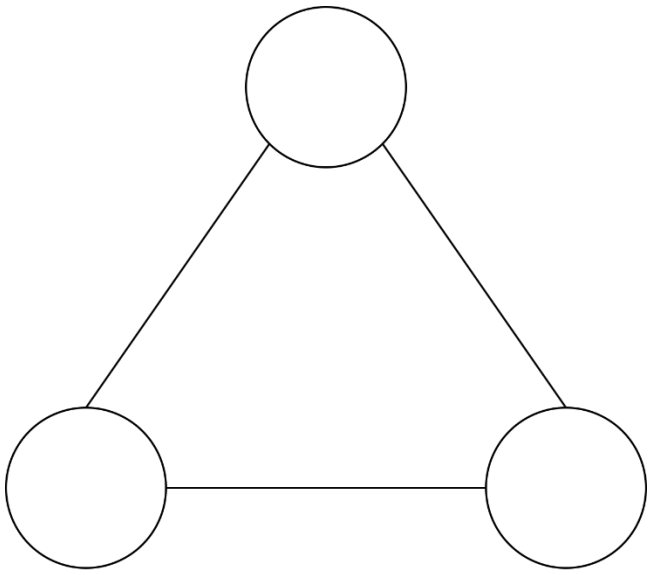
   (a) Koita keksiä/löytää algoritmi joka tarkastaa onko graafissa yhtään siltaa ja mahdollisesti palauttaa yhden (tai useamman) tällaisen kaaren. Voit käyttää apunasi lähteitä jos mainitset ne.

   (b) Anna esimerkki graafista jossa ei ole yhtään siltaa

   (c) Miksi kutsutaan graafia jonka jokainen kaari on silta?

a) Tarjanin algoritmi on algoritmi, jolla voidaan löytää kaikki sillat graafista. Se on modifikaatio DFS algoritmista, missä pidetään kirjaa siitä, milloin solmu on löydetty ja mikä olisi aikaisin mahdollisin aika milloin solmu voitaisiin löytää.
Lähde:
https://www.geeksforgeeks.org/bridge-in-a-graph/

b) Graafi, jossa ei ole yhtään siltaa



c) Graafia, jossa jokainen kaari on silta, kutsutaan puuksi.