TKT20001 Tietorakenteet ja algoritmit (syksy 2017)

Kurssikoe 2 (19.12.2017)

Tentissä **saa** olla mukana yksi **käsin kirjoitettu** kaksipuolinen A4-kokoinen "lunttilappu".

Vastaa kuhunkin tehtävään **erilliselle** konseptipaperille. Kirjoita jokaisen paperin yläreunaan **kurssin nimi, päivämäärä, nimi, nimikirjoitus ja opiskelijanumero.** Vaikka jättäisit johonkin tehtävään vastaamatta, palauta silti vastauspaperi kyseiseen tehtävään.

Lunttilappua, kysymyspaperia ja **suttupapereita ei** palauteta.

Tehtävissä, joissa pyydetään koodia tai päätät vastata koodilla, voit käyttää luentojen (Cormenin) tyyppistä pseudokoodia tai muuta ymmärrettävää pseudokoodityyliä / ohjelmointikieltä, esim. Javaa. Jos käytät oikeaa ohjelmointikieltä, selitä erityisen hyvin, mitä ohjelmassasi tapahtuu, äläkä käytä mitään kielen erikoista piirrettä tai valmiita kirjastoja.

Vastaa kaikkien kysymysten kaikkiin kohtiin. Kokeen maksimipistemäärä on 22.

1. [*4 pistettä*] Vastaa lyhyesti jokaiseen alakohtaan.

   (a) Mitä järkeä on *kekojärjestämisessä* kun pikajärjestäminen on yleensä nopeampi ja lomitusjärjestäminen vakaa?

   (b) Toimivatko seuraavat algoritmit negatiivisilla kaaripainoilla? Perustele.

   - *BFS*
   - *Bellman-Ford*
   - *Dijkstra*
   - *Kruskal*

   Lauseen tai kahden vastaus joka kohtaan riittää.

   (c) Minkälaisiin verkkoihin *Ford-Fulkeron* menetelmää voi soveltaa. Ja mitä menetelmää soveltamalla saadaan aikaan.

2. [*6 pistettä*]

   (a) Kuvaile lyhyesti miten voit tarkastaa onko annettu *suunnattu* verkko DAG, eli *suunnattu syklitön* verkko. Kauanko tähän kuluu aikaa?

   (b) Entä miten tarkastetaan onko annettu *suunnattu* verkko *vahvasti yhtenäinen*? Ja aikavaativuus?

   (c) Voiko DAG olla *vahvasti yhtenäinen.* Todista.

Käännä!

3. [*6 pistettä*] Uolevin setä haluaa paeta vankilasta. (Kannattaisi seurata rakennusmääräyksiä eikä rakentaa mielivaltaisen korkuisia aitoja.)

   Uolevi haluaa auttaa setäänsä. Käytettävissään Uolevilla on vankilan pohjapiirrustus ja vanginvartioiden aikataulut. Uolevi haluaa algoritmin joka saa syötteenään vankilan pohjapiirrustuksen ja pakoon käytettävissä olevan ajan sekunteina, ja laskee näiden perusteella onko pako mahdollinen vai ei.

   Pohjapiirrustus syötetään kaksiulotteisena taulukkona johon on näppärästi merkitty kuinka kauan uolevin sedällä kestää edetä kunkin kohdan läpi. **Esim.** seuraavassa kartassa uolevin sedän koppi on kokonaan nollia koska setä voi aloittaa pakonsa mistä tahansa kohdasta omaa koppiaan. Sellissä on jostain kumman syystä 2 ovea joiden tiirikointiin menee 5 ja 6 sekuntia. Käytävällä liikkuminen on nopeaa (1 sekunti per ruutu), ja ulko-ovien/ikkunoiden läpäisyyn vaaditaan 7 tai 8 sekuntia. Seinät on merkitty #-merkillä koska ei liene mielekästä olettaa että paon aikana ehtii murtaa seiniä.

```
######7###
8111111118
##6#1##6##
#00#1#111#
#00515111#
##########
```

Nyt siis jos Uolevin toivomaa algoritmia kutsuttaisiin yllä olevalla pohjapiirrustuksella ja ajalla 20, tulisi algoritmin palauttaa `True`, koska setä voi paeta Tiirikoimalla pohjoisen oven 6, juoksemalla käytävän länsipäätyyn +2 ja puristautumalla ikkunan kaltereiden välistä +8. Eli yhteensä 16 sekuntia.

   (a) Suunnittele Uoleville tehokas algoritmi. Erityisesti Uolevi toivoo että algoritmi tekisi mahdollisimman vähän turhaa työtä jos pakoyritys on tuhoon tuomittu. Ota myös huomioon että Uolevin setä on hieman outo eikä osaa liikkua viistosti.

   (b) Mikä on algoritmisi aikavaativuus suhteessa *kartan kokoon*? Entä suhteessa *pakoon käytettävään aikaan*? Voit tarvittaessa olettaa että pienin mahdollinen siirtymän arvo on 1.

4. [*6 pistettä*]

   (a) Mitä voit päätellä verkon pienimmästä ja suurimmasta virittävästä puusta jos verkossa on yksi komponentti ja $|E| = |V| - 1$ kaarta.

   (b) Mitä seuraava koodi tekee?

```
f(G, k)
1  for all v ∈ V
2      make-set(v)
3  järjestä kaaret (u, v) ∈ E painon mukaan
4  while joukkojen määrä > k
5      valitse seuraava kevein kaari (u, v) ∈ E
6      if find(u) ≠ find(v)
7          union(find(u), find(v))
8  return { (v, find(v)) | v ∈ V }
```

   (c) Voiko saman tehdä myös *Primin* algoritmilla? Miten tai miksei?