Obligatorisk Oppgave 4

Algoritmer og Datastrukturer ITF20006

Lars Vidar Magnusson

Frist 28.04.15

Den tredje obligatoriske oppgaven tar for seg forelesning 9 til 13, som dreier seg om datastrukturer.

Praktisk Informasjon

Det er med disse oppgavene dere skal få den nødvendige erfaringen dere trenger, så det kreves at alle innleveringer skal være individuelt arbeid. Det er selvsagt til lov å hjelpe hverandre, men den endelige besvarelsen skal være et resultat av eget arbeid. Besvarelser som viser tegn plagiering vil bli underkjente.

Dere kan selv velge i hvilket språk dere implementerer algoritmene, men hjelpefunksjoner vil bli gitt i Java. Hvis dere skulle trenge direkte hjelp med koden, kan det være greit å holde seg til enten Java eller C#.

Dere skal lage en enkel rapport som beskriver besvarelsen og samler alle tekstlige svar. Rapporten skal struktureres i henhold til oppgaven. Koden som løser programmeringsoppgavene skal være kjørbar og skal ta en inputfil som argument fra kommandolinjen, for så å skrive ut løsningen. Koden skal leveres i ren kildekode (ikke et prosjekt fra en eller annen IDE med en masse unødvendige filer), og det skal legges ved et enkelt skript som kompilerer koden.

Både rapport, kode og skript skal pakkes i en tarball og skal sendes til meg på lars.v.magnusson@hiof.no med emne "ITF20006 $OPPGAVE $STUDENT".

Oppgave 1 - Implementere Dijkstra

I denne oppgaven skal dere implementere Dijkstra algoritmen for å finne korteste vei mellom to noder i en urettet graf. Dere kan ta i bruk kode som er lagt ut på fagsiden, men det er ikke lov å benytte seg av hverken standard eller tredjeparts bibliotek. Pseudokoden for algoritmen er listet under.

```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V          // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Algoritmen benytter seg av en prioritetskø og en hjelpefunksjon RELAX. En implementasjon av en prioritetskø er lagt ut på fagsiden. Pseudokoden for RELAX algoritmen er listet under.

```
RELAX(u, v, w)
  if v.d > u.d + w(u, v)
      v.d = u.d + w(u, v)
      v.π = u
```

Dere skal bruke et heltall for å identifisere hver node. For enkelhets skyld kan dere anta hvis vi har n noder i en graf så er de identifisert av $\{0, 1, \ldots, n-1\}$. Man kan da enkelt bygge en graf ved å først angi antall noder, så angi alle kantene i grafen ved å spesifisere de to nodene hver kant kobler sammen.

Dere står fritt i om dere vil representere en graf ved hjelp av en nabomatrise eller ved hjelp av nabolister. En nabomatrise er nok muligens lettere å implementere, men den tar opp unødvendig mye plass hvis grafen inneholder relativt få kanter.

Resultatet av et søk skal skrives ut til skjerm. Hvis det eksisterer en sti mellom to noder skal først nodene så den totale lengden skrives ut i en mellomromseparert liste. Hvis det ikke finnes en sti mellom de to nodene skal **NA** skrives ut.

Dere må også lage kode for å lese inn en graf fra en fil. Den første linjen i en slik fil inneholder et heltall som angir hvor mange noder det er i grafen. De resterende linjene inneholder kantene i grafen. Hver linje har tre heltall, de to første angir hvilke noder det går en kant mellom, det siste angir vekten til kanten.

Oppgave 2 - Kjøring av Dijkstra

I denne oppgaven skal dere teste implementasjonen dere laget i forrige oppgave. Dere skal måle både kjøretid og minneforbruk. Når det er snakk om kjøretid kan det være nyttig å skille mellom tid brukt til å bygge en graf og tiden det

tar å finne korteste vei mellom to noder. Dere skal teste med flere ulike inputstørrelser og søkenoder og rapportere gjennomsnittstiden for søkene på de ulike inputstørrelsene. Resultatene skal diskuteres i forhold til både teorien i kurset og implementasjonsdetaljene.

Lykke til!