**Oblig 2 – IN2010**

**Redegjørelse for oppgavene:**

Oppgave 1)

* Vi begynte å kode oppgave 1 med formål om å representere grafen på en objekt-orientert måte, men det visste deg fort at vi lagde en kombinasjon av en objekt-orientert implementering og en «adjacency list». Dette fordi både skuespillere filmer har egne klasser de blir representer. Skuespillere har også en liste over filmer de spiller i og en liste med kanter som fører til dens naboer. Det samme gjelder filmer, som holder på en liste over skuespillere som har spilt i den.
* I tillegg til at skuespillere(noder) holder på en liste med kanter som leder andre skuespillere som har spilt i samme film, har vi en egen ArrayList som holder på alle kanter fra grafen.
* For å genere grafen, må man kalle på disse metodene i «main» i rekkefølgen de er oppgitt her:

1. generateMoviesNActors()
2. generateEdges()

Oppgave 2)

* I oppgave 2 har vi valgt BFS-algoritmen for å traversere gjennom grafen, da vi mener den er enklere å implementere for en så tett graf. Når vi prøvde å implementere DFS, ble resultatet en uendelig output stream. Denne feilen skyldes bruk av feil vilkår i en av løkkene.
* Kjørtidskompleksiteten til BFS er O(|V| + |E|). Grunnet feil i «movies.tsv» filen, har vi ikke muligheten til å kommentere den eksakte kjøretiden. Det skyldes at vi må bruke en egen funksjon innebygd i VS Code til å kjøre programmet.
* For å finne den korteste veien og vise den som output i terminalen, vi først hente de to ønskede nodene vi skal finne veien mellom, så kalle på metoden shortestPathBetween() der de nodene vi har hentet blir parametere.

Oppgave 3)

* I oppgave tre benytter Dijsktra-algoritmen for å finne den mest underholdende veien mellom to skuespillere i grafen. Dette fordi Dijsktra har en kjøretid på O(E \* log(V)) som er raskere en Bellman-Ford med kjøretid O(V\*E). En annen grunn til at vi velger Dijsktra, er at vi ikke trenger å forholde oss til negative kanter. Vi kunne i teorien brukt Prim også, men siden måten vi implementerer Dijkstra minner om måten vi bruker BFS på, har vi latt være å endre noe.
* For å finne den mest underholdende veien og skrive den ut i terminalen må vi kalle på metoden ChillestPathBetween på samme måte som metoden shortestPathBetween i oppgave 2.

Oppgave 4)

* I oppgave 4 bruker vi igjen BFS til å telle komponenter og fjerne fra en kopi av alle arraylisten med alle skuespillerene. Da vet vi nøyaktig hvilke noder som ikke er en del av en komponent og kjører derfra BFS algoritmen på de resterende nodene. Finner så ut hvilke komponenter vi har og størrelsen på dem. Kaller på opg4 for å finne disse. Alt gjøres i Oblig2.java