aronbj@uio.no taeriks@uio.no

IN2010 – Innleveringsoppgave 3

Instruksjoner for bruk innleveringsoppgave 3:

I utgangspunktet trenger man kun å kjøre Python filen innleveringsoppgave3, slik man ellers ville forventet.

Kommentarer angående kjøretid og implementasjon av algoritmene:

Vi har implementert flere algoritmer for å løse oppgaven, og tenker derfor at det er hensiktsmessig å foreta en kort kjøretidsanalyse og forklaring av dem. For å strukturere selve grafen har vi valgt en struktur der grafen består av to typer noder, altså filmer og skuespillere. Filmer har kanter til skuespillere, men skuespillere har ingen kanter seg imellom. Film objekter har heller ingen kanter seg imellom, derav bindes de sammen av skuespillere som har flere filmer tilfelles. For å fremstille node objektene i grafen har vi implementert klassene Movie og Actor. Foruten om dette er det verdt å nevne at antallet noder og kanter i grafen er annerledes enn fasiten for oppgaven. Det er fordi jeg har valgt en struktur med to nodetyper. Dette øker det totale antallet noder i grafen, men senker antallet kanter betraktelig.

Den første prosedyren som benyttes er byggGrafen(). Her utføres det noen operasjoner som er på formen O(n), slik som fil-innlesing for filmer, med noen konstanttidskjøreoperasjoner som tilordninger i lister og ordbøker. Deretter er det to større kodesnutter, den ene for fil-innlesing og tilordning av skuespillere og den siste for å sette filmer og skuespillere sammen i grafen. Fil-innlesingen for skuespillere mener vi at er på formen O(n + k) ettersom vi her gjør en operasjon som itererer først gjennom hver linje i filen, og for hver linje i filen over alle elementene over index 2. For den siste kodesnutten mener vi også at det foregår på O(n^2) ettersom vi her går igjennom hver nøkkel i en overordnet ordbok med hver film og skuespiller. Deretter går man igjennom alle verdiene til hver skuespiller og film, for så å tilordne de som kanter for nøkkelen med tilhørende vekter.

Deretter har vi implementert en Merge Sort algoritme, som benyttes avslutningsvis for å enklere systematisere og telle komponenter i grafen vår. Den er på O(n log(n)), den deler opp et array rekursivt for så å sette det sammen på sortert form fra lavest til størst.

Deretter har vi implementer ulike algoritmer som bygger på BFS, bfs_shortest_paths_from() og bfs_shortest_path_between(). Implementasjonene er BFS med korteste stier til alle noder fra en enkeltstående node, og en metode som finner korteste sti mellom to noder. Kjøretiden skal her være O(|V| + |E|).

Det er også en utskriftsmetode kalt utskrift() som går på O(n), for å gjøre utskriften på graftreversering noe penere.

Det er to implementasjoner av en dijkstra algoritme. Altså, shortest_paths_from() og dikstra_shortest_path_between(). Disse skal ha kjøretidskompleksistet O(|E| + log(|E|).

Til slutt har det vært implementert et BFS søk for å telle antall komponenter i grafen vår, bfs_komponenter() for å traversere grafen, og komponenter() for utskrift. bfs_komponenter() antar vi at forekommer på kjøretiden O(V + E) for hver komponent i grafen. Det itereres ikke over en node som er besøkt fra før flere ganger. Deretter er det en

operasjon som utføres på formen O(n) for å telle og skrive ut antallet forekomster av komponenter med visse størrelser.

Potensielle feil og mangler:

Det er mulig at selve grafbyggingen kunne vært gjort kjappere, samt at implementasjonen av ulike metoder for utskrift, kan ha vært overflødige. Det hadde kanskje vært mulig å heller benytte str() mer aktivt for de ulike nodetypene.

Min opplevelse av kjøretiden har variert mellom de ulike enhetene som jeg har testet programmet på. Fint om jeg kan få tilbakemelding om hvordan programmet presterte relativt til forventingene.