LØSNINGSSKISSE TIL EKSAMENSOPPGAVE I FAG TDT4305 – MAI 2017

NB! Dette er ikkje fullstendige løysingar på oppgåvene, kun skisse med viktige element, hovudsakleg laga for at vi skal ha oversikt over arbeidsmengda på eksamen, og som huskeliste under sensurering. Det er også viktig å være klar over at det også kan vere andre svar ein dei som er gjeve i skissa som vert rekna som korrekt om ein har god grunngjeving.

Oppgåve 1 – Hadoop – 15 % (alle delar tel likt)

a) Blokker av fast størrelse replikert til *n* maskiner, fortrinnsvis på meir enn eitt rack. Viktig poeng at granulariteten er blokk og ikkje fil. Forøvrig, sjå pensum for meir detaljar.
b) Handsamar namnerom til filsystem (filer, katalogar, og metadata for desse).
   Informasjon lagra persistent på lokal disk som to filer: namespace image og edit log.
   Også oversikt for kvar fil kvar blokker er lagra.
   Ikkje nødvendig å lagre dette persistent (kan rekonstruerast frå datanodar ved restart).
c) Data representert som key-value-par
   Map (k1, v1) → List(k2, v2)
   Reduce (k2, List(v2)) → List(k3, v3)
   Viktig å få med at output er liste, mange bommar her.

Oppgåve 2 –Spark – 20 % (alle deler tel likt)

a) val data = sc.textFile("countries.tsv").
b) val data2 = data.map(x => x.split("\t"))
c) val data3 = data2.map(x => (x(0), x(1).split(",")))
d) data3.flatMap(x => x._2).count() eller
   data3.flatMapValues(x => x).count()
e) data3.flatMap(x => x._2).distinct().count()
   /(latMapValues ville ikke virke her siden output er (Land,Term))
f) val data4=data3.flatMapValues(x => x).distinct().map(x => (x._1, 1)).reduceByKey((x,y)=> x+y).collect()
   (Ikkje trekk for manglande collect)

Oppgåve 3 – NoSQL – 5 %

Consistency – er kopiar like?
Availability – alltid tilgjengeleg
Partition tolerance - kan systemet køyre sjølv om det er partisjonert?
CAP-teorem: Ikkje mogleg å tilby alle tre samtidig i eit distribuert system med replikering
Forventa forklaring som viser forståing av termane (og ikkje berre term på C/A/P).

Oppgåve 4 – MinHashing – 10 % (alle deler tel likt)

a) Shingles er n-gram, kontinuerlige subsekvensar av token, som kan brukes for å måle grad av likhet mellom dokument.

b) Svar:

	S1	S2	S3	S4
h1	2	1	1	0
h2	0	0	0	0

Oppgave 5 – Datastrømmer (streaming data) – 30 % (alle deler teller likt)

Anta at du er ansatt i et firma som analyserer interessetrender for det nye kameraet Sony A9 som Amazon nettopp har startet å selge. Firmaet ditt har bestemt seg for å gjøre dette basert på sosiale media data, inkl. Twitter og Facebook.

a) Gå ut fra at du starter med å bestemme deg for finne antall meldinger som nevner ”Sony A9” ved å bruke såkalte stående spørring (”Standing Query”). Forklar hva som er forskjellen(e) mellom ”standing query” og ”ad-hoc query”. Hvordan kan ”standing query” løse oppgaven?

Svar:

- **Ad-hoc queries**: Normale spørringer som kjøres en gang.
  Feks.: Hva er maks. verdi sett så langt?
- **Standing queries**: Spørringer som i prinsippet kjøres på strømmen hele tiden.
  Feks.: Rapporter hver gang man ser en ny maks verdi i datastrømmen.
- For å løse oppgaven kan vi bruke spørring som kjøres på datastrømmen hele tiden slik at for hver ny melding som inneholder ”Sony A9” skal man returnere en ny verdi på S, hvor S er summen av antall meldinger som inneholder ”Sony A9” i datastrømmen.

b) Siden vi kan se sosiale media data som datastrøm har de også de andre karakteristikker og/eller utfordringer enn statiske data. Drøft hva disse er.

Svar: Her kreves det at man lister minst tre karakteristikker samt konsekvensene av disse (utfordringene) for å få full pott.
Karakteristikker:

- Data kommer hele tiden, dvs. uendelig og uten grenser
- Har varierende format og struktur, for eksempel bilder, tekst, etc.
- Kommer i høy hastighet og er dynamisk
- Har stort volum

Utfordringer ligger i å ha metoder for å håndtere disse i form av for eksempel plass (minne og lagringsplass), tid og hastighet (inkl. sanntidsprosessering og skalerbarhet).

c) Som alternativ til ”standing query” velger du ”bit counting” for løse deler av oppgaven. Forklar hvordan kan oppgaven oversettes til ”bit counting”.

Svar: Bit counting kan løse oppgaven ved å oversette alle meldinger til binære tall, hvor en bit settes til 1 dersom meldingen inneholder ”Sony A9”, 0 ellers. Vi reduserer da problemet til å telle enerne i bitstrømmen ved for eksempel å bruke DGIM algoritmen.
NB: For å få full pott holder ikke bare å si at man teller enerne men også nevne hvordan dette gjøres i praksis.

d) Analysen din skal se på trendene de siste 2 ukene og du ser for deg antall ”klikk” og ”kjøp” av produkter i Amazon direkte som en del av analysen. Se for deg at du skal finne ut hvor stor andel av ”klikk” og ”kjøp” er gjort på ”Sony A9”. Til dette formålet velger du nå å bruke glidende-vindu-prinsippet (“sliding window”). Anta at dette vinduet har en størrelse på 500 ”klikk” og ”kjøp” tilsammen. Forklar hvordan du går fram for å beregne denne andelen.

Svar: Her kan man fortsatt bruke bit counting som utgangspunkt. Deretter forventes at man forklarer hvordan man beregner andelen av meldinger som inneholder ”Sony A9” innenfor et vindu med størrelse 500 bits (i/500). Videre er det nødvendig med oppdatering av denne andelen for hver ny som melding kommer inn.

e) Når vi først er inne på ”klikk” drøft hvordan ”bloom filter” kan være nyttig til å finne antall klikk. Gjør de antakelsene du finner nødvendig.

Svar: Bloom filter kan brukes her til å “filtrere” bort gjentakende klikking fra same person, dvs. man kan bruke bloom filter som et steg i tellingen av antall unike klikk. Dvs. først kan man lage et bloom filter ved for eksempel hashe på både URL (linken som klikkes på) og bruker ID (feks. IP-adressen til brukeren). Dette vil hjelpe oss i å identifisere om man har sett klikkingen til brukeren før. Hvis en klikk *ikke er sett* før økes en teller. Ellers lar man telleren være uendret.

Hovedpoenget med spørsmålet er å finne ut om man har forstått prinsippet med Bloom Filter.

f) Bruk “bloom filter”-prinsippet til å fylle ut tabellen nedenfor

Strømelement	**Hash-funksjon - h_1**	**Hash-funksjon - h_2**	**Filtrere Innhold**
			00000000000
56 = 11 1000	110=6	100=4	00001010000
428 = 1 1010 1100	10010=18 mod 11 =7	1110=14 mod 11 =3	00011011000
875 = 11 0110 1011	10111=23 mod 11=1	11001=25 mod 11 =3	01011011000

Hint: bruk *h1(x)=y1 mod 11* og *h2(x)=y2 mod 11* som hash-funksjoner, der verdiene av *y1 og y2* er hentet henholdsvis fra oddetalls-bits fra *x* og partalls-bits fra *x*.

Oppgave 6 – Anbefalingssystem og sosiale grafer (recommender systems and social graphs) – 20 % (alle deler teller likt)

Firmaet du er nyansatt i er spesialist på anbefalingssystemer. Din oppgave er å utvikle gode anbefalingsalgoritmer og metoder.

a) Med fokus på fordeler og ulemper sammenlikn ”collaborative filtering” mot ”content-based recommendation”.

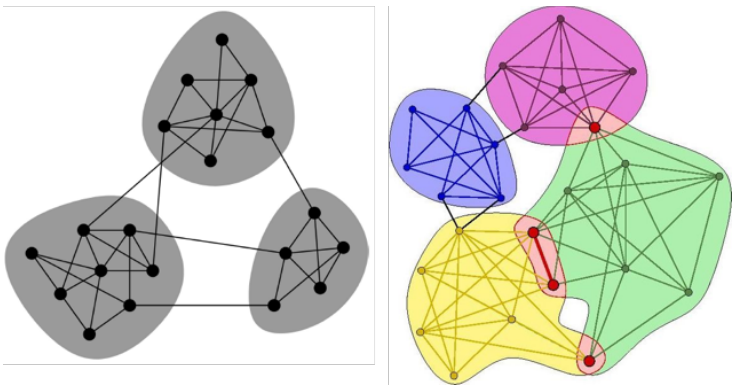
Svar:

Collaborative Filtering	**Content-based Recommender**
Fordeler: - Fungerer på alle type produkt - Lett og intuitivt å sette opp basert bruker/produkt-matrisen	**Fordeler**: - Trenger ikke data på andre brukere - Kan gi anbefaling til brukere med unike smak - Kan anbefale nye og ikke populære produkt (ingen cold start på nye produkter) - Bakgrunnen for anbefaling kan forklares til bruker
Ulemper: - Cold-start-problemet: trenger nok brukere i systemet til å finne ”match” - Sparsitet: veldig få produkt har ratings. Problemer med å finne nok bruker-ratings på produkt - First raters: Kan ikke anbefale produkt som ikke har ratings fra før - Popularitets bias: Kan ikke anbefale produkt til brukere med unike smak. Mest tendens til å anbefale populære produkt.	**Ulemper**: - Avhengig av god feature engineering, i.e., utfordrende med å hente ut features på alt - Anbefaling til nye brukere er avhengig av å kunne ha en god brukerprofil - Overspesialisering, i.e., anbefalingene har tendens til å være kun innen brukerens profil (lav grad av serendipity)

b) Anta at du vil bruke bruker-relasjoner, som feks. “friends” og “followers” fra sosiale media, til å bygge opp brukerprofiler. Slike relasjoner kan tegnes i en graf. Bruk dette som utgangspunkt til å svare på følgende spørsmål:

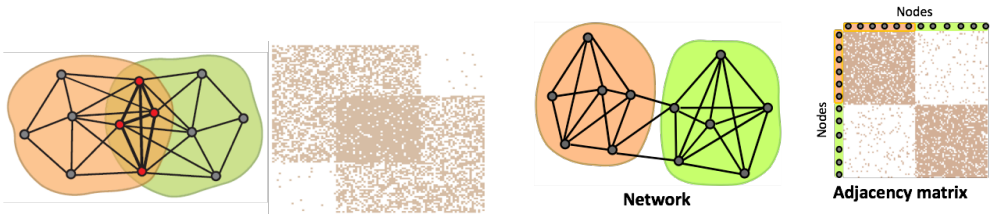
i. Forklar forskjellene på ”*overlapping*” og ”*non-overlapping communities”* i grafer. Hvordan kan vi enkelt finne ut om to grafer overlapper.

Svar:



Overlappende vs. ikke overlappende som vist på figuren over har vi dersom en eller flere noder kan identifiseres til å tilhøre flere “communities”. En enkel måte å finne dette ut på er å bruke

naboskapsmatrise (adjacency matrix). Jo mer kant-tetthet i denne matrisen jo større sannsynliget for overlapp (se figurene nedenfor).



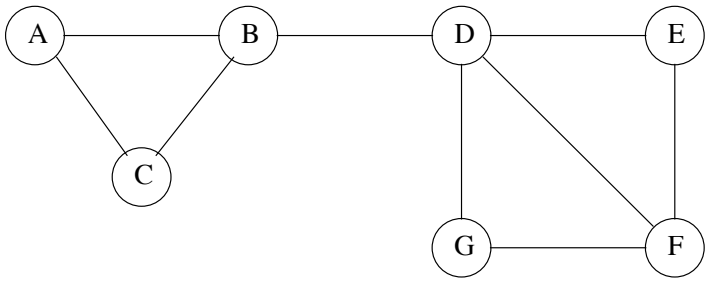
(For å få full uttelling holder ikke bare å nevne at en eller flere noder hører til i flere communities men også hvordan man finner ut dette ved for eksempel overnevnte matrise).

ii. Hva er hovedhensiktene med "community detection"? Forklar.

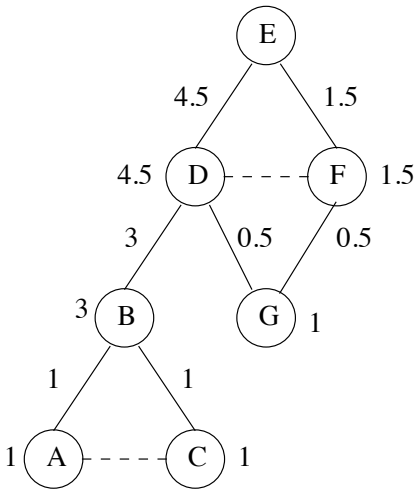
Svar: Hovedhensiktene til "community detection":

- Community detection er en form for graf-clustering som brukes til å oppdage grupperinger og relasjoner mellom grupper.
- Å kunne analysere trender
- Å kunne identifisere grupper som igjen kan brukes til enten markedsføring eller produktanbefaling.

c) Girvan-Newmans metode for beregning av "betweenness" er en vanlig metode innen graf-mining-teori. Vis hvordan du går fra Figur 1 til Figur 2 nedenfor ved å bruke Girvan-Newmans metode til å beregne "*betweenness*" i grafen i Figur 1.



Figur 1: Start-graf

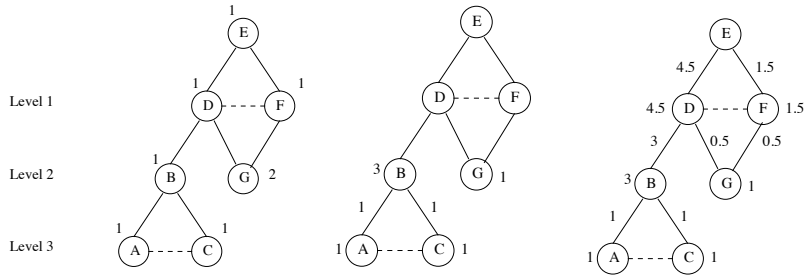


Figur 2: Steg 3 i første iterasjon.

Svar:

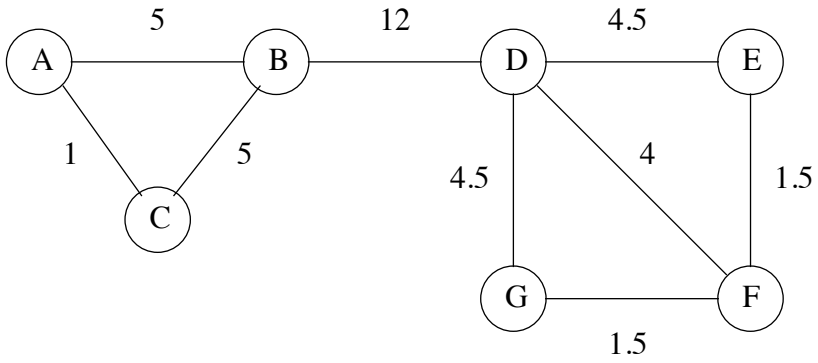
Vi bruker BFS som beskrevet i læreboka.

Gjøre om E til rot-node.

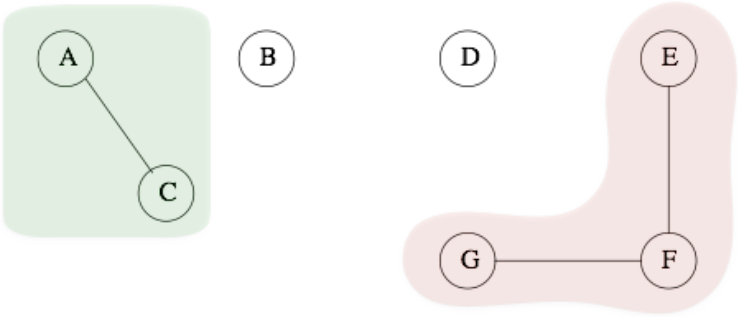


Steg 1 Steg 2 Steg 3

d) Hvordan kan resultatfiguren nedenfor brukes til detektere ”communities”?



Svar:
Vi fjerner alle kanter med **høyste betweeness-verdi** og sitter til slutt igjen med ”communities” som da er komponenter som koplet sammen.



Herfra kan vi også få hierarki av communities.