Andy Mateo

Project 2 (HetioNet) - Building off Project 1 implementation for wider use and more queries.

Query 1: For each drug, compute the number of genes and the number of diseases associated with the drug. Output results with top 5 number of genes in a descending order.

Design Pattern

1. Filtering and Mapping
   - Start by filtering out the edges to include only the relevant metaedges that represent relationships between drugs and diseases or drugs and genes
   - There will be two maps, one map that's filtered by CbG, CdG, and CuG, and the other that's filtered by CpD and CtD. The end result is one map with key-value pairs (Compound, Gene), and another map with key-value pairs (Compound, Disease)

2. Summarization
   - Count the number of genes and diseases associated to a drug by filtering out the distinct (Compound, Gene) and (Compound, Disease) pairs and then mapping each compound in the drug-gene map and drug-disease map to a key-value pair of (Compound, 1)
   - Aggregate the counts for the drug-gene list and drug-disease list using reducebyKey()

3. Joining
   - Combine our two maps using a fullOuterJoin(). The result is map with key value pairs (Compound, (Gene Count, Drug Count))
   - Now set Gene Count to be either the gene count for the compound or 0, and Drug Count to be either the disease count for the compound or 0
   - We need to use fullOuterJoin() and do the second step because if we just used join(), the resulting map would only contain compounds found in both the drug-gene list and drug-disease list. For those compounds not shared by the two maps, we need to set the default Gene Count and Drug Count to 0.

4. Sorting
   - Our current map is of the form (Compound, (Gene Count, Drug Count)). Use a map to turn it into the form of (Compound, Gene Count, Drug Count).
   - Finally, sort our list by descending order based on the Gene Count, using takeOrdered() to get the top 5

Pseudocode:

```
class MAPPER_1:
    method MAP (string source, string metaedge, string target)
        if metaedge ∈ {“CbG”, “CdG”, “CuG”} then
            EMIT (source, target) // (drug, gene) pair
        if metaedge ∈ {“CpD”, “CtD”} then
            EMIT (source, target) // (drug, disease) pair

class REDUCER_1:
    method REDUCE (drug d, genes [gene1, gene2, …])
        unique_genes <- SET (genes)
        EMIT (d, COUNT(unique_genes)) // (drug, gene_count) pair

class REDUCER_2:
    method REDUCE (drug d, diseases[disease1, disease2, …])
        unique_diseases <- SET (diseases)
        EMIT (d, COUNT(unique_diseases)) // (drug, disease_count) pair

class MAPPER_2:
    method MAP (drug d, (gene_count gc, disease_count dc)) // joined gene and disease
count map
        gene_count <- gc if not null else 0
        disease_count <- dc if not null else 0
        EMIT (d, gc, dc) // (drug, gene_count, disease_count) tuple

class SORTER:
    method SORT(results)
        sorted <- SORT_DESCENDING (results, key = gene_count)
        return TOP(sorted, 5) // sorted list of top 5 based on gene_count
```

Result:

```
[('Compound::DB08865', 580, 1),
('Compound::DB01254', 558, 1),
('Compound::DB00997', 528, 17),
('Compound::DB00390', 521, 2),
('Compound::DB00170', 521, 0)]
```

Query 2: Compute the number of diseases associated with 1, 2, 3, …, n drugs. Output results with the top 5 number of diseases in a descending order.

Design Pattern

1. Filtering and Mapping
   - Start by filtering out the edges to include only the relevant metaedges that represent relationships between between drugs and diseases (CpD, CtD)
   - Flip the key-value pairs so that the resulting map is a list in the form of (Disease, Compound)

2. First Summarization
   - Count the number of drugs associated to a disease by filtering out the distinct (Disease, Compound) pairs and then mapping each disease in the map to a key-value pair of (Disease, 1)
   - Aggregate the count using reducebyKey() so that our resulting map has key value pairs (Disease, Drug Association Count)

3. Second Summarization
   - Count the number of diseases associated with n drugs by mapping each drug association count in our map to a key-value pair of (Drug Association Count, 1)
   - Aggregate the count using reduceByKey() so that our resulting map has key value pairs (Drug Association Count, Number of Diseases Associated with Drug Association Count)

4. Sorting
   - Finally, sort our list by descending order based on the Number of Diseases Associated with Drug Association Count, using takeOrdered() to get the top 5

Pseudocode:

```
class MAPPER:
    method MAP (string source, string metaedge, string target)
        if metaedge ∈ {“CpD”, “CtD”} then
            EMIT (target, source) // (Disease, Drug) pair

class REDUCER_1:
    method REDUCE (disease d, drugs [drug1, drug2, …])
        unique_drugs <- SET(drugs)
        EMIT (d, COUNT(unique_drugs)) // (Disease, Drug Association Count) pair

class REDUCER_2:
    method REDUCE (drug_count dc, disease [disease1, disease2, …])
        EMIT(dc, COUNT(disease)) // (Drug Association Count, Disease Count) pair

class SORTER:
    method SORT (results)
        sorted <- SORT_DESCENDING (results, key = disease_count)
        return TOP(sorted, 5) // sorted list of top 5 based on disease_count
```

Result:

```
[(1, 10),
(2, 7),
(3, 6),
(11, 6),
(9, 6)]
```

Query 3: Get the name of drugs that have the top 5 number of genes. Output the results.

Design Pattern

1. Filtering and Mapping
   - Start by filtering out our node map so that it only contains compounds
   - Create a new map with key value pairs (ID, Name)

2. Joining
   - Using the result that we got from Query 1, join our top five list with this new map using join(). The end result is a map with key-value pairs (ID, (Gene Count, Name))
   - Change the map so that it contains key-value pairs (Gene Count, Name)

3. Sorting
   - Finally, sort our list by descending order based on the Gene Count, using takeOrdered() to get the top 5

Pseudocode:

```
class MAPPER_1:
    method MAP (string id, string name, string kind) // input line from nodes.tsv
        if kind = ‘Compound’ then
            EMIT (id, name) // (drug_id, drug_name) pair

class MAPPER_2:
    method MAP (drug_id, gene_count, disease_count) // result from Query 1
        EMIT (drug_id, gene_count)

class MAPPER_3:
    method MAP (drug_id, (gene_count, drug_name)) // joined gene_count and name maps
        EMIT (name, gene_count)

class SORTER:
    method SORT (results)
        sorted <- SORT_DESCENDING (results, key = gene_count)
        return TOP (sorted, 5) // sorted list of top 5 based on gene_count
```

Result:

```
[('Crizotinib', 580),
('Dasatinib', 558),
('Doxorubicin', 528),
('Digoxin', 521),
('Menadione', 521)]
```