APPROACH

The approach is based on Agglomerative Clustering strategy. I referred Simple HAC algorithm from Manning's book on Information Retrieval (Chapter 17). First step is to create the Similarity matrix by calculating similarity scores between the documents. I have calculated the similarity using cosine similarity. The TF-IDF scores are used to build the similarity scores. Each document is compared among all other possible pairs of documents including itself and similarity is assigned using dot product between the two.

Let D1 and D2 be the two documents, then:

Cosine Similarity (D1, D2) = D1.D2 / |D1| |D2|

Once the similarity matrix is built, clustering algorithm starts and continues until all the possible clusters have been formed with no pair having similarity > 0.4

Simple HAC Algorithm:

1. Build the Similarity Matrix of the whole corpus.

- Iterate over all possible pair of documents in pair.
- Evaluate cosine similarity of each pair and store them.

2. Run Clustering Algorithm

- Initiate with all documents as individual clusters
- Find out the most similar clusters, i.e. pair with highest similarity score (say, cluster_x and cluster_y).
- Merge the clusters – cluster_x and cluster_y, into a new cluster- cluster_xy (According to Single-Linkage Strategy)
- Remove cluster – cluster_x and cluster_y from the stored collection
- Add a new cluster cluster_xy into the collection
- Store the newly formed cluster in a data structure
- Run the clustering algorithm tills stopping is reached

3. Display all the clusters created (stored in array list) order-wise.

DATA STRUCTURES USED:

Similarity Matrix:

Similarity Matrix is maintained in the form of HashMap. Key of this Map is name of cluster (String), and its value is a Priority Queue ordered in descending order of the similarity scores. An inner class is created to store document id and its similarity score corresponding to the cluster (key of map). A custom Comparator is built to maintain the required order of scores in a priority queue.

To keep it simple, a cluster is named as per documents in the cluster. It's a comma separated string with the corresponding documents. Like, cluster containing documents 102, 105 and 110 would be:

102.html, 105.html, 110.html.

Building Similarity Matrix:

Two for loops are used for building similarity matrix, to find out each pair and compute similarity. One document is picked (say 1.html). Its similarity is evaluated with all the other documents in corpus. Their scores are stored in a dictionary ordered by decreasing sequence of corresponding score. A nested dictionary is used as data structure.

So, the key of the map would be 1.html. And the value would be another dictionary containing all other documents and corresponding similarity scores.

Let's say 7.html is most similar to 1.html, 10html is second most similar, and 27.html is least similar. Then Map would look like this.

Key = 1.html

Value = (7.html, 0.93); (10.html,0.89) …… (27.html, 0.02)

```
for i in all_txt_files:
    documentSimilarity(i,i)  # Document Similarity between the same document
    for j in all_txt_files:
        documentSimilarity(i, j)  # i and j are the first two documents
```

Finding Most Similar and Dissimilar Documents:

As per agglomerative clustering, each document forms an individual cluster. First two documents that merge and form a cluster would be the most similar ones. As the output suggests, 417.html and 420.html are the most similar ones. It turns out that these documents are same and contain same terms.

To find most dissimilar documents, the cosine similarity of the two should be minimum across corpus. Following approach is used to find them:

1. Iterate over key-value set of Similarity Matrix. Maintain the global minimum score and documents.
2. Find the value set corresponding to the current key.
3. Now find the cluster which has minimum similarity in the dictionary. As dictionary is ordered according to similarity scores, the last element would be the one with lowest score. So, in simple words, get the last element of the dictionary.
4. Compare the similarity queue with global minimum, if found lesser, update the global minimum documents.
5. Repeat steps 2 to 4 for each key in step 1.
6. Global minimum html docs would be the most dissimilar

In the corpus, the most dissimilar documents are: 484.html and 99.html.There could be (in fact there are) more such pairs which are dissimilar whose similarity scores are equal with the mentioned pair (0 score). This are just one of these dissimilar pairs

Output Format:

The program prints the clusters formed in an order line-by-line. The name of new clusters formed will be comma separated string of document names present in the cluster.

Example: Following are the first few lines of my output. See comments in line 16 of below output for clarification

1. 1st cluster: 417.html; 2nd cluster: 420.html
2. 1st cluster: 102.html; 2nd cluster: 130.html
3. 1st cluster: 422.html; 2nd cluster: 424.html
4. 1st cluster: 421.html; 2nd cluster: 423.html
5. 1st cluster: 416.html; 2nd cluster: 418.html
6. 1st cluster: 412.html; 2nd cluster: 414.html
7. 1st cluster: 403.html; 2nd cluster: 405.html
8. 1st cluster: 404.html; 2nd cluster: 406.html
9. 1st cluster: 413.html; 2nd cluster: 415.html
10. 1st cluster: 399.html; 2nd cluster: 400.html
11. 1st cluster: 429.html; 2nd cluster: 431.html
12. 1st cluster: 407.html; 2nd cluster: 410.html
13. 1st cluster: 409.html; 2nd cluster: 411.html
14. 1st cluster: 438.html; 2nd cluster: 442.html
15. 1st cluster: 430.html; 2nd cluster: 432.html
16. 1st cluster: 413.html, 415.html; 2nd cluster: 417.html, 420.html