Day-2

==========

1. Submit a single python file which contains implementation of all the task functionalities — <Roll_No>_D2.py
2. The input to the program will come from a file containing the source document — input.txt, and the final output has to be written in 'Summary_SentenceGraph.txt'.

==========

If you carefully observe the sentences in the output summary of PageRank, you will see that some sentences might convey redundant information. To get rid of the redundancy, we use two approaches: a) Maximal Marginal Relevance scheme to rerank sentences while introducing diversity, and b) a sentence-clustering followed by a word-graph based approach.

Using MMR to omit 'redundancy' in the summary

To make the output more diverse, we introduce the concept of MMR (Maximal Marginal Relevance) which is expressed below.

$$MMR = Argmax_{D_i \in R \setminus S}[\lambda * Imp(D_i) - (1 - \lambda) * max_{D_j \in S} Sim(D_i, D_j)]$$

MMR is in essence a reranking strategy which penalizes sentences that are scored higher but are semantically similar, thus introducing diversity in the generated output.

So to get rid of redundancy, after obtaining PageRank scores for a sentence, rerank the sentences using MMR and then print top-n sentences as the summary. Ensure that the ordering of sentences is the same as that in the document.

Clustering and Word-Graph approach

T1. Sentence clustering using K-Means

1. Obtain the tf-idf representations of the sentences using the algorithm described in T2 (of Day 1) with preprocessing as described in T1 (of Day 1).
2. Apply K-means clustering on the tf-idf representation on the sentences, and obtain K clusters of sentences as follows.

```
1. Choose the number of clusters(K)
2. Place the centroids c_1, c_2, ..... c_k randomly
   (use random.sample utility in python for this)
3. Repeat steps 4 and 5 until convergence — cluster assignment does
   not change
      4. for each sentence i:
            - find the nearest centroid(c_1, c_2 .. c_k) —using
      cosine similarity on tf-idf representation
            - assign the sentence to that cluster
      5. for each cluster j = 1..k
            - new centroid = mean of all points in that cluster
```

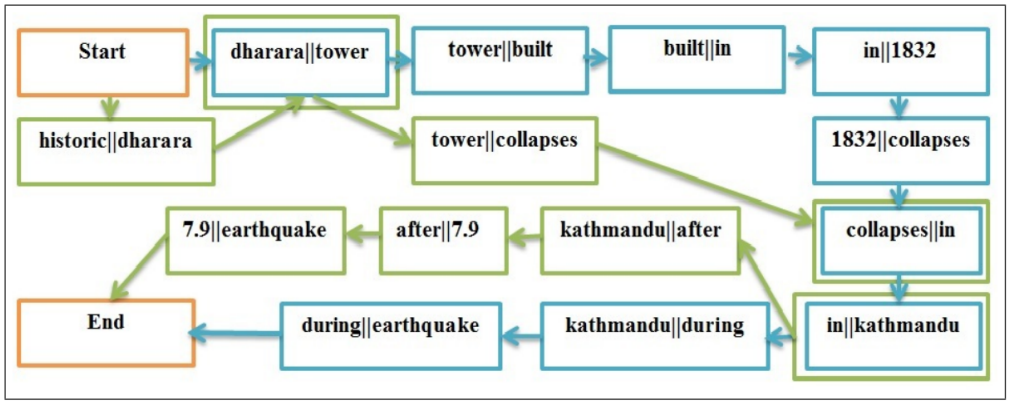
T2. Obtaining 'new' sentences using Sentence Graph

1. For each of the clusters,
    a. identify the sentence that is closest to the cluster centroid. In case of ties, pick any one at random — **S1**
    b. Find a sentence in the cluster that has at least 3 bigrams in common with S1 — **S2**.

c. In case no **S2** exists for a cluster — save only **S1** from this step and return. Do not continue with Step d. and part 2.
d. Construct a sentence graph **G** using S1 and S2 as explained below:
    i. Take a dummy start node and a dummy end node and insert it into the graph **G**
    ii. Obtain the bigrams for the sentence — *l_bigram*
    iii. For each bigram b_i in *l_bigram :*
        if b_i is not in G : Add a node for this bigram in G
        If b_i is first bigram:
          add an incoming edge from the dummy start node to b_i
        else If b in the last bigram:
          add an outbound edge from b_i to dummy end node
        else:
          add an incoming edge to b_i from b_[i-1]
    iv. Do steps ii and iii for both S1 and S2

2. Generate a sentence by traversing from dummy start node to dummy end node through a path chosen randomly.

Eg, Consider two sentences — (i) dharara tower built in 1832 collapses in kathmandu during earthquake, and (ii) historic dharara tower collapses in kathmandu after 7.9 earthquake. The word-graph is shown below.



A new sentence that can be constructed from these two sentences is: ***dharara tower built in 1832 collapses in kathmandu after 7.9 earthquake.***

These newly generated sentences are now the sentences we consider as summary sentences. The only issue is the ordering between the sentences, which we solve in the next task.

T3. Identifying the ordering of the clusters

In order to identify the ordering among the sentences, we apply a simple ordering procedure.

1. For each cluster sentence, identify the index of **S1 (**from T2**)** in the original document. The index of **S1** will be the order of the cluster.
2. Arrange the sentences generated from each of the clusters (in **T2)** in the increasing order of the cluster_order as obtained in step 1.

For example, if there are three clusters c1, c2, and c3:
If c1's **S1** is at index (4) — ordering of c1 is 4,
If c2's **S1** is at index (0) — ordering of c2 is 0, and
If c3's **S1** is at index (10) — ordering of c3 is 10.

Let Summ1, Summ2, and Summ3 be the cluster sentences obtained from c1, c2, and c3. Then the final summary will be: **<Summ2>. <Summ1>. <Summ3>.**

Using this approach identify the arrangement of sentences and output the arranged sentences as the generated summary.