Assignment 3
Distributed Systems, Monsoon 2017
Deadline: 3rd October. 2017 at 9 PM.

Note: The process of evaluation will be automated, and hence, you are expected to stick to the input and output format strictly. We will give you a script, and some sample data, so that you can ensure that your code doesn't violate the expected ip/op format.

Q1. Your task is to construct a **'term document matrix' (TD)** using MPI in C++. Term frequency (tf) is an Information Retrieval concept that tells us how "important" a word is to a corpus. A simple metric of the importance of a word to the document is its word count. However, this should be normalised to the size of the document. Hence, you are expected to use "double normalization" with a factor (K) of 0.5 (Check Wikipedia). A TD matrix is a collection of term frequencies for a bunch of documents.

double normalization K	$K + (1-K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

"Stop Words" are words which are filtered out before or after the processing of Natural Language data. They are usually the most commonly occurring words such as "the", "a", "it" etc. that skews the TD Matrix. Since they are irrelevant in the task of retrieving the words which are important to a document, they should be removed from your answer. As there isn't any standard list of stop words, we expect you to use the "Default English Stopwords" list from http://www.ranks.nl/stopwords.

For the purpose of automation, *both* the words and documents should be listed in a *lexicographic order*. The output should be invariant to uppercase/lowercase characters. The input is a list of books, *in any order*, and the output is a list of words with their frequencies in each of the books.

Sample Run:

```
$>> mpirun -np 5 td_matrix
goblet_of_fire.txt a_game_of_thrones.txt

a_game_of_thrones.txt
arya 509
cersei 1041
fire 479
potter 0
ron 0
snow 1011
```

stark 5424

goblet_of_fire.txt
arya 0
cersei 0
fire 161
potter 6231
ron 5343
snow 0
stark 0

Explanation: Note that each document contains a list of all the words in the dictionary. While some words are unique to each book (Eg: “Stark” to “a_game_of_thrones” and “potter” to “goblet_of_fire”) and does not exist in the others, there can be words that are common to both the documents (such as “fire” occurring 479 times in “a_game_of_thrones” and 161 times in “goblet_of_fire”)

Note:
1. Your code should be independent to the no. of processes used at runtime.
2. You can use any NLP library to parse the documents. They can be used ONLY for removing the punctuation and converting the document into words. However, this conversion should be done in chunks, by each of the child process. *In other words, you should parallelize even the parsing of the document.*
3. While you have to use MPI with C++, you can use Python for parsing the document, and call it using os.system(part_of_the_document) from C++ in each of the child processes. Make sure to ensure parallelism.
4. You can use the Gutenberg Dataset to test your code (https://www.gutenberg.org/)

Q2. Write an MPI program that implements **Merge Sort**.

- The master process divides the user input array into ***n*** equal parts and sends them accordingly to the slave processes.
- The slave processes use any sorting algorithm and sends back the sorted subarray to the master.
- Upon receiving all the sorted subarrays from the slave processes, the master merges the subarrays into one and prints them out.

Ex: mpiexec -np 5 MS
Input: 9 5 1 2 4 7 5 3

Master to Slaves:
[9, 5] will be sent to Slave 0.
[1, 2] will be sent to Slave 1.
[4, 7] will be sent to Slave 2.
[5, 3] will be sent to Slave 3.

Slaves to Master:
[5, 9] is received from Slave 0.
[1, 2] is received from Slave 1.
[4, 7] is received from Slave 2.
[5, 3] is received from Slave 3.

Master merges the subarrays and prints the following:
1 2 3 4 5 5 7 9

Note:
1. Print only the final sorted array.
2. There would be checks ensuring two way communication between Master and Slaves. Submitting a simple sorting program will result in negative marks.

Q3. Write an MPI program to implement **Minimal Spanning Tree (MST)** of a weighted, undirected graph.

The input to the program should be the no of vertices and Edges and for each edge ,
Weight 'w' should be given.

Input : V (no of vertices) E(no of edges)
Start-vertex end-vertex   edge-weight

```
Ex: 4 5
    1 2 4
    1 3 6
    1 4 5
    2 3 3
    3 4 7
```

Each process is assigned a subset of vertices and in each step of computation, every process finds a candidate minimum-weight edge connecting one of its vertices to MST. The root process collects those candidates and selects one with minimum weight which it adds to MST and broadcasts result to other processes. This step is repeated until every vertex is in MST.

Output : The output should consist of the edges(both vertices) which are in the Minimal Spanning Tree and their weights. Each edge should be in a new line.
For the above example, the output should be -

```
1 2 4
1 4 5
2 3 3
```

Q4. **Distributed Grep :-**

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

The aim is to perform a parallel search on a textfile using MPI with the MapReduce paradigm.

Master Process divides the input file into N chunks and sends one to each slave process.

Map Stage :-

A slave process will read a chunk line by line (separated by '\n') and find occurrences of the search term. The search can be performed using STL or built in libraries. It will return these lines to the master process.

Reduce Stage :-

The master process will count the number of lines that matched. It then prints all the matched lines and the total count.

Print the matched lines in the same order in which they occur in the file.

inputfile.txt

Lorem Ipsum is simply dummy text of the printing and typesetting industry.
It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged.
Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

Ex: mpiexec -np N inputfile.txt Lorem

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.
total=2

Upload Format:

1. Roll No. (Folder)
    a. Q1_<Roll no>.cpp
    b. Q1_parse.py (if there)
    c. Q2_<Roll no>.cpp
    d. Q3_<Roll no>.cpp
    e. Q4_<Roll no>.cpp
    f. Readme.pdf

Marking Scheme:

- Each question has an equal weightage of 90 marks for the implementation and 10 marks for the VIVA. The total marks are 400.
- Your code shall be given marks proportional to the test cases it passes. In case it doesn't follow the ip/op format specified, it shall be automatically evaluated to be 0.
- At the time of manual evaluation, only your VIVA shall be taken.
