Code Explanation.

**Part1:**

First, we take source files from the user using the command line argument and then retrieve all the details in it. Source file consists of different engine queries and their runs, this data is in a format such that each engine and their query runs are separated with a new line. Each line consist of engine name its query run and retrieved results produced by that engine and related results for that particular query all these data are separated with ';' hence we read source file and first we separate each engine by splitting data by '\n' which separate each engine data. After that we extract each engine query run data by splitting data with ';' hence we will get engine name, query run, retrieve results and related results.

example:- sourcefile.txt



Suppose the above file is an input file.

we scan this file using the scanner and store it in a variable e.g. String input\_file = scanned\_File

now we split it using string function split() and store it into an array

-> String[] engines\_data = input\_file.split("\n");

now engines\_data array have values of all these engines in array

suppose we want 1st value what we will do is we call -> String engineA = engine\_data[0];

similarly, we get all values by using for loop which runs from 0 to length of an array.

now let see how to extract each engine's data like query run results etc.

suppose we got value when for loop run 1st time ( 1;A;RNNRUNNRURRRUNUNNNNRR;10 )

now we want to extract engine name, query run, retrieve results and related results, etc.

we split above data using ->String[] engine = engine\_A.split(";");

now engine array consist query to run an array 0, engine name at array 1, retrieve results at array 2, related results at 3

Hence,

Query\_run = engine[0];// i.e. 1

Engine\_name = engine[1]; i.e. A

Retreived\_results = engine[2]; i.e. RNNRUNNRURRRUNUNNNNRR

Related\_results = engine[3]; i.e. 10

Since we got all these data we now calculate precision, recall, etc.

for precision at each recall point

* count how many 'R' present in retrieved result using charAt("R") function of String class.
* calculate total character count till that particular 'R'.
* calculate precision and recall at each recall point
* store these data in the array list

now divide R count with total count till particular position we will get precision at recall.

also, calculate recalls when we reach at 'R' by counting the loop and dividing it by rel\_ret

when loop complete we will get total precision and total recall

for precision at a particular point (e.g. p@5)

we call array list e.g. for precision at 5 -> precision\_at.get(4)// array list starts with 0

for p@r=0.5 we calculate it as follow

for 1st query of engine A related result value is '10'.

recall\_point/rel\_ret = 0.5

recall\_point = 0.5 x rel\_ret

recall\_point = 0.5 x 10

recall\_point = 5

now we will query array list for precision at 5th recall point precision\_at\_recall.get(4)

for the inverted index, we create a HashMap(inverted\_index) and put 11 recall value from 0 to 100

we have two array list for precision array list is 'precisions' and for recall array list is 'recalls'

now we check if precision values at recall points are present in array list then put it into inverted\_index HashMap

save all these data in part1output.txt file

**Part2:**

We take two command-line arguments 1st file contains document results with score and engine name or number and 2nd file contain engine list and their weight. We read both the file using Scanner and store them into String variables named results and engines respectively.

Suppose file containing results is results.txt and file containing engines detail is engines.txt

result and engine txt file data look like this



As seen from the above image of results.txt each engine retrieve results are in the column and each result consist of engine name, document number and document score separated by ';'

We extract all these data as follows

* store text file data in any String variable say results
* split results values by '\n' and store splatted data in a result\_line String array
* split each data of result\_line array by ';' and store data in result\_details array

Consider 1st value (A;26;400) here engine name is at 1st position, the document number is at 2nd position and document score is at 3rd position and this data is stored in result\_details array

Hence we get these details from the array as follow

* engine = result\_details[0]//at 1st position we get engine name, array start from 0
* document\_number = result\_details[1]//at 2nd position of array we get document no.
* rank\_score = result\_details[3]//at 3rd position we get document score

Similarly, we obtained engine name and weight from engine.txt file

we split data by '\t' and then further split it by ';' to get required data

store engine name and weight in engine\_weight\_list HashMap for further use

Normalization:-

suppose engine A has scored for some document 1.4 and engine B has to score of 900 for the same document. in such a scenario we can't add these scores hence we need to normalize them.

for normalization, we required a minimum and maximum document score for each engine

while extracting all the data from a text file when we extract document score we also store it into 'min' HashMap and 'max' HashMap so that we can use it during normalization

Interleaving:-

for interleaving, we just have to add documents in round-robin fashion and just to make sure document don't repeat as they can be present in different engine results we first check if document already present in 'interleaving\_list' array List or not if it's not present already then add it else don't add it now just iterate over this array list and print top 100 documents obtained by interleaving

Comb Sum:-

for comb sum, we need to normalize the results of every engine as each engine has a different mechanism to assign a score to documents. for normalization we required a formula which is => (doc\_score -min score)/(max score - min score).

we have doc score obtained from result.txt file we also have minimum and a maximum score of engine store in HashMap’s 'min' and 'max' respectively. Hence by applying the above formula, we get a normalized score.

since we have got a normalized score now we perform a comb sum in which we have to get all same document present in any engine and add there score and store in comb Sum HashMap.

Now copy comb Sum HashMap value into the 'comb\_sum\_list' array list and sort it in descending order by comparing there score.

Lcm:-

We use the normalized score in lcm also. we also need engine weight which will be multiplied by score rest of the operation is the same as comb sum.

we create 'lcm' HashMap and store values of lcm in it for each document

copy lcm HashMap value into 'lcm\_list' array List and sort it in descending order by comparing documents rank score.

**Part3:**

Part3 consists of two types of data engine retrieved result and live data obtained from sourcefile.txt and live\_data.txt like files.

We have to make probfuse model for which we take sector size from the command line argument and make probfuse model using engine data and apply it to live data

Engine results are like given in below image



We obtained engine name, query run, retrieve results and related results as explained in part1

let say the user has given k value 3, now we have to divide retrieved results into 3 parts

similarly, if the user is given k value 4 we have to divide retrieved results into 4 parts

suppose retrieve results for engine A query run 1 is (RNNRUNNRURRRUNUNNNNRR)

and k is 3 then the above result will be divided as follow (total number of result/k) i.e 21/3 = 7.

results will be divided and the size of each segment is 7 (RNNRUNN) - (RURRRUN) - (UNNNNRR)

now we calculate precision for each segment.

let consider 1st segment (RNNRUNN) => total retrieve results are 7 and the number of 'R' is 2

hence precision = 2/7. similarly, we calculate precision for remaining segments and store them in a HashMap named as 'model'.

take an average of k value of queries of a particular engine

now add all k precision values of each engine in 'add\_k' HashMap.

once probfuse model is ready we will apply it to live data



* above is an example of live data in which engine name with their retrieved documents is shown.
* we use a scanner to read this file and store it into a string variable
* then we use the same method as documented in part1 to extract data like engine name and their documents.

Apply probfuse model to this data by dividing live data into the number of segments which will be calculated using k value.

now using probfuse model we give the score to documents and store into live\_data\_map which is a HashMap after that copy HashMap values into array List and sort them into descending order by comparing their score.