1. Corpus
2. Pre process the corpus
3. ~~Stemming~~
4. Lemmatization
5. Stop word removal
6. Posting list
7. Inverted index - positional index
8. Search for query
9. Ranking
10. Response time
11. Efficiency

Give DocId to URLs

**Doubts:**

Documents → URLs or CSV files

Corpus static or dynamic

UI

TASK 1: create dictionary docid , term, term frequency, position

1.1. Preprocess the snippet

Url ------ Snippet

To

Url ------ Processed Snippet

Add another column to dataset

1.2. Read all csv

T-tid = {}

d= { term\_id1: {

docs: { d1: { pos: {r1: [p1,p2], r2:[p1,p2]}, term\_freq=F1},

d2:{pos: [p1,p2..], term\_freq=F2},

……….. }

doc\_freq:f1 //number(d1+d2+...)

},

term\_id2:{

docs: { d1: { pos:[p1,p2], term\_freq=F1},

d2:{pos: [p1,p2..], term\_freq=F2},

……….. }

doc\_freq:f1 // <F1+F2+...>

}}

Index compression

**TASKS:**

~~TASK 1: index creation : sriya~~

TASK 2: spelling correction (tolerant query): Abhishek

~~TASK 3: Search the query : lavanya~~

TASK 4: Ranking: bhavna

Abhishek:

1. Phonetic spelling correction
2. contextual spelling correction
3. elastic search comparison

TODO:

Results comparison with elastic search

Performance comparison with elastic search

Performance measurement mechanism(evaluation metrics, time(query response time), F1 score etc)

Measure engine in some way

Additional functionality for extra marks:

~~Wild card query~~

~~More than one index~~

Word error correction in query (phonetic)

coding some performance metric with a set of evaluation data

Commonly used biwords(indexing)

$ query.py

>> Date: “ ” (= < >)

>> Show: “ single term Q“

>> Snippet: “multi term query Q”