Problem Statement

Develop a simple Information Retrieval System using Boolean Retrieval model. Following are the functionalities to be implemented.

- Read the content of the documents
- Create Inverted Index
- Implements positional index Algorithm
- Create appropriate interfaces for inputting the query, value of k and display the query results.

Technical Specifications

The algorithms for inverted index and positional index is implemented in *Python*.

The user interface is achieved simply by using *HTML5* and *CSS3*.

The UI is connected to the server side language using *Flask* which is a microframework for python based on *Werkzeug*, *Jinja2* and good intentions.

Algorithms Implemented

- **Inverted Index**: In inverted index, we maintain a dictionary of terms. Then for each term, we have a list that records which documents the term occurs in. Each item in the list is conventionally called a *posting*. The list is then called a postings list (or inverted list). All the postings lists taken together are referred to as the postings.

Building an Inverted Index involves following steps:

1. Collect the documents to be indexed.
2. Tokenize the text, turning each document into a list of tokens.
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms.
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

The following is the algorithm for querying using an inverted index:

```
def queryInvIndex(s):
    answer = []
    #arr = re.compile('[&|]|\sand\s|\sor\s').split(s)
    #arr = re.split('(\W)', s)
    fn = set(fileNames)
    arr = s.strip().lower().split()
    flag = 1
    for ind in range(1,len(arr),2):
        if not(arr[ind]=="&" or arr[ind]=="|" or arr[ind]=="and" or arr[ind]=="or"):
            flag = 0
    if flag==0:
        print "Invalid Expression."
    else:
        for ind in range(0,len(arr),2):
            if '~' in arr[ind]:
                answer.append(fn - set(terms[arr[ind][2:-1]].keys()))
            else:
                answer.append(set(terms[arr[ind]].keys()))

        x = 1
        for ind in range(1,len(arr),2):
            if arr[ind] == "&" or arr[ind] == "and":
                answer[0] = answer[0] & answer[x]
            elif arr[ind] == "|" or arr[ind] == "or":
                answer[0] = answer[0] | answer[x]
            x+=1

        return list(answer[0])
```

- **Positional Index:** For each term, we store the position(s) in which tokens of it appear.

  <**term**, number of docs containing **term**;
  doc1: position1, position2 … ;

  doc2: position1, position2 … ;
  etc.>

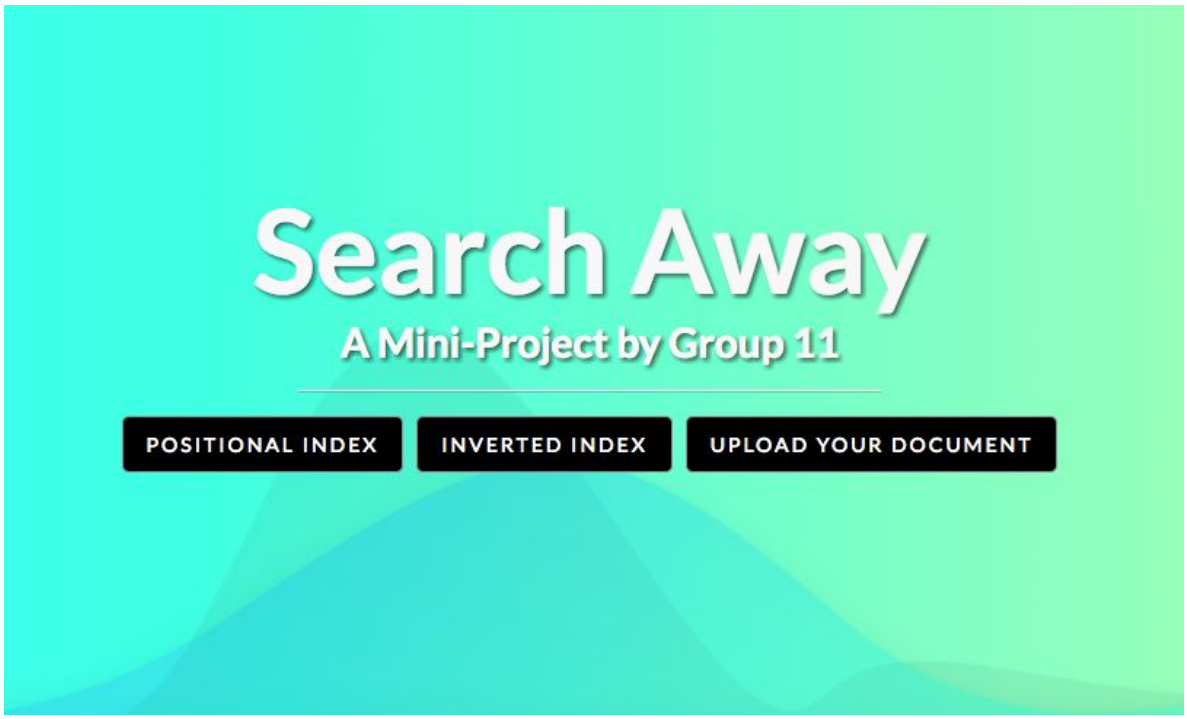
  To process a phrase query, we still need to access the inverted index entries for each distinct term.
  We start with the least frequent term and then work to further restrict the list of possible candidates. In the merge operation, the same general technique is used as before, but rather than simply checking that both terms are in a document, we also need to check that their positions of appearance in the document are compatible with the phrase query being evaluated. This requires working out offsets between the words.

  Following is the algorithm for positional index.

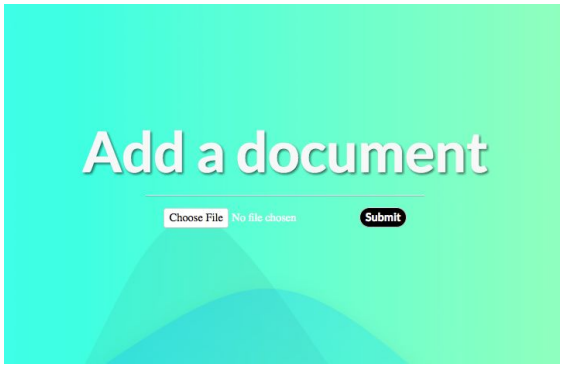
```
def positionalIntersect(p1,p2,k):
    answer = []
    intersection = list((set(p1.keys())) & (set(p2.keys())))
    for x in intersection:
        pos1 = p1[x]
        pos2 = p2[x]
        for y in pos1:
            for z in pos2:
                if abs(y-z)<=k:
                    answer.append((x,[y,z]))

    return answer
```

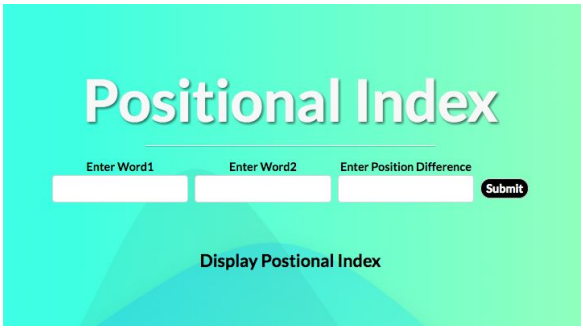
User-Interface

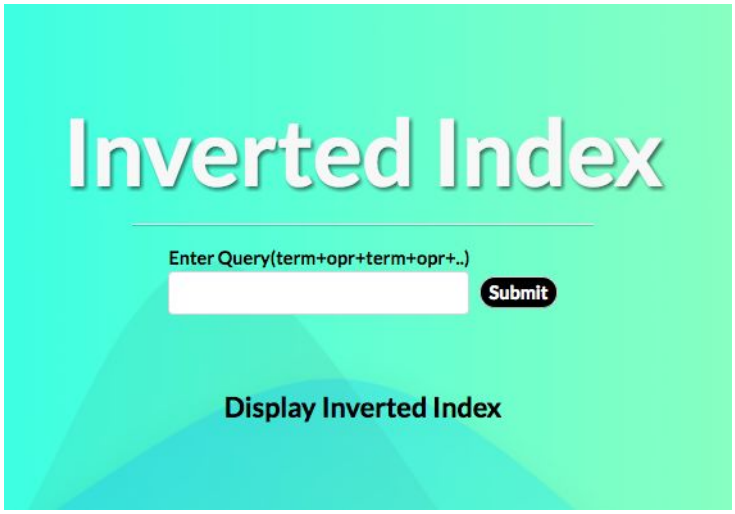
Home page:



Other Features:







Conclusion

This is a micro model showing the potential of what a properly executed Information Retrieval System can do like Google, in which the principles of algorithms are developed to match interests of the user to the information available best about those interests.

Inverted index and Positional index though are a basic functionality in an information retrieval system; they form the foundation of any good one.