IR Assignment-1

1. Preprocessing

```
def preprocessing(d):
    no_punc = []
    d_lower=d.lower()

    nltk_tokens = nltk.word_tokenize(d_lower)

    stop_words_removed = []
    for w in nltk_tokens:
        if w not in stop_words:
            stop_words_removed.append(w)


    new_words = []
    for x in stop_words_removed:
        if(x.isalnum() and x!=" "):
            new_words.append(x)


    return new_words
```

For preprocessing of data:
We have applied the following steps-

1. Converting to lower case
2. Converting into tokens
3. Removing the stopwords
4. Removing the special characters

2. **Creating the posting dictionary and implementing the unigram inverted index data structure.**

```
id = 0
postings = {}
mapping = {}
file_path_list = []

for fileName in os.listdir():
  path = "/Users/ektagambhir/Downloads/Humor,Hist,Media,Food 2" + '/' + fileName
  file = open(path, encoding ="ascii", errors="surrogateescape")
  current = file.read().replace('\n', ' ')

  mapping[fileID] = fileName
  file_path_list.append(path+ '/' + fileName)

  tokens = preprocessing(current)

  for i in tokens:
    if i in postings:
      postings[i].append(id)
    else:
      postings[i] = [id]

  id = id+1
```

For creating the posting list we have tokenized each document and checked the document ID the respective token is present in.
The posting dictionary consists of words and the corresponding list containing the doc ids.

3. Providing Support for various operations-

```
def boolean_queries(query,operations):
    op=[]
    if(query[0] in postings.keys()):
        op = postings[query[0]]
    for i in range(1,len(query)-1):

            if(query[i] in postings.keys()):
                a=query[i]
            else:
                a=[]
            a1=set(a)
            output=set(op)
            if operations[i-1]=='OR':
                output.union(a1)
            elif operations[i-1]=='AND':
                output.intersection(output)
            elif operations[i-1]=='OR NOT':
                atemp=output.difference(a1)
                output.union(atemp)
            elif operations[i-1]=='AND NOT':
                atemp=output.difference(a1)
                output.intersection(atemp)
            else:
                print("Wrong operation")
    return len(output);
```

This function will return the length of the query result for the given input.

- For Providing the support for “OR”, we have used the union function of sets.
- For Providing the support for “AND”, we have used the intersection function of sets.
- For Providing the support for “OR NOT”, we have first done the “difference” function of the set and then used the union function.
- For Providing the support for “AND NOT”, we have used the “difference” function of sets and then used the intersection function.

4. Calculating results and finding the number of comparisons on the input by the user.

Merge Functions-

This function will find the common elements on 2 sorted lists (list of doc ids contained after performing user-entered operations) and finds the number of comparisons.

```
comparisons=0;
def merge_and(ans1,ans2):
    res = []
    i = 0 ; j = 0
    comparisons = 0
    while i < len(ans1) and j < len(ans2):

        if ans1[i] == ans2[j]:
            res.append(ans1[i])
            i += 1
            j += 1
            comparisons += 1
        elif ans1[i] < ans2[j] :
            i += 1
            comparisons += 1
        else :
            j += 1
            comparisons += 1
    return res, comparisons
```

The following function will count the number of elements on union of both the lists. And returns the no. of comparisons.

```
def merge_or(post_list1, post_list2):
    comparisons=0;
    res = []
    ptr1 = 0 ; ptr2 = 0
    while ptr1 < len(post_list1) and ptr2 < len(post_list2):

        if post_list1[ptr1] == post_list2[ptr2]:
            res.append(post_list1[ptr1])
            ptr1 += 1
            ptr2 += 1
            comparisons += 1
        elif post_list1[ptr1] < post_list2[ptr2] :
            res.append(post_list1[ptr1])
            ptr1 += 1
            comparisons += 1
        else :
            res.append(post_list2[ptr2])
            ptr2 += 1
            comparisons += 1


    while(ptr1 < len(post_list1)):
        res.append(post_list1[ptr1])
        ptr1 += 1

    while(ptr2 < len(post_list2)):
        res.append(post_list2[ptr2])
        ptr2 += 1

    return res, comparisons
```

At last, the following function is called and following are returned:

1. Number of documents matched
2. Number of comparisons done
3. Name of the documents (document list)

```
def ansD():
    query=takeinquery();
    operations=takeinoperators();
    queries(query,operations)
    num_docs_match,num_of_comp,doc_list= queries(query,operations)
```

Q2 :

Preprocessing :

```
def preprocessing(d):
    no_punc = []
    d_lower=d.lower()

    nltk_tokens = nltk.word_tokenize(d_lower)

    stop_words_removed = []
    for w in nltk_tokens:
        if w not in stop_words:
            stop_words_removed.append(w)


    new_words = []
    for x in stop_words_removed:
        if(x.isalnum() and x!=" "):
            new_words.append(x)


    return new_words
```

In preprocessing steps data from each file is taken and passed in function called preprocessing , at first as it can be seen above data is changed to lowercase , tokenization is then done using nltk library . Using the list of stopwords retrieved from NLTK corpus stop words are removed from data and stored in a list ie stop_words_removed . From this list we are removing punctuations and other symbols , empty tokens are also removed.

Building positional index :

```
def positional_indexing(dict_position , file_id  , data):

    for position , token in enumerate(data):


        if token in dict_position:
            dict_position[token][0] +=1
            if file_id in dict_position[token][1]:
                dict_position[token][1][file_id].append(position)

            else:
                dict_position[token][1][file_id] = [position]

        else:
            new_dict = {}
            new_dict[file_id] = [position]
            dict_position[token] = [1 , new_dict]


    return dict_position
```

Here in this step a dictionary is passed in this function , along with file index and data(preprocessed) in it , now for each token in this data positional index is being created , if the word is already present in dictionary for a particular file then its position is appended to already existing list in dictionary ,else a new list is created.

Phrase searching:

```
def search_phrase_query(query):

    posting_ds = load_datastructure()
    no_phrase=0
    flag = 0
    postings = []
    for word in query :
        if word in posting_ds:
            postings.append(posting_ds[word])
        else:
            flag=1
            break

    alldocs = []
    if(flag==1):
        print("1")
        print("Given phrase is not present")
    else:
        for i in range(len(postings)-1):
            j=i+1
            for j in range(len(postings)):
                docs = []
                x = j-i
                docs = intersection(postings[i], postings[j] , x)
                if(len(docs)!=0):
                    alldocs.append(docs)
                else:
                    print("2")
                    print("Given phrase is not present")
                    no_phrase=1
                    break
            if no_phrase == 1:
                break

        if no_phrase == 0:
            final_result = set.intersection(*[set(x) for x in alldocs])
            end_result = []
            end_result = list(final_result)
            if(len(end_result) == 0):
                print("3")
                print("Given phrase is not present")
            else:
                print("Number of documents retrieved:")
                print(len(end_result))
                print("Names of documents are:")
                for f in end_result:
                    print(file_names[f])

                return end_result
        else:
            end_result="nothing"
            return end_result
```

This function handles searching a phrase in given documents , here a query is passed as a parameter after preprocessing it and postings all pairs of tokens along with index difference is passed in intersection function which returns those docs where these tokens are present and at same distance , this process is repeated for each pair of tokens and intersection of those documents is taken to find final list of documents in which that phrase is present.

Intersection :

```
def intersection(posting1 , posting2 , diff):
#     len1 = len(posting1[1].keys())
#     len2 = len(posting2[1].keys())
    len1 = posting1[0]
    len2 = posting2[0]
    doc_ids = []
    #print("here")
    swap_flag=0
    if(len1<=len2):
        ps1 = posting1
        ps2 = posting2
        #diff=1
        #print("-1")
    else:
        ps1 = posting2
        ps2 = posting1
        diff = -diff
        #diff=-1
        swap_flag=1
        #print("0")

    for file_id in ps1[1].keys():
        if file_id in ps2[1].keys():
            #print("1")

            for pos1 in ps1[1][file_id]:
                #print("1.1")
                for pos2 in ps2[1][file_id]:
                    #print("1.2")
                    if pos2-pos1== diff :
                        if file_id not in doc_ids:
                            #print("2")
                            doc_ids.append(file_id)
                        break

    return doc_ids
```

This intersection function is used to find those documents where particular tokens are present and also at same distance as present in that query , if for some document if both are not present or distance between them does’nt match with distance in that query then that document is discarded.

Output:

```
query = input("Please enter your query")
query = preprocessing(query)
#query = perform_stemming(query)
#print(query)
if(len(query)==1):
    posting_ds = load_datastructure()
    word = query[0]
    if word in posting_ds:
        d = posting_ds[word][1].keys()
        d = list(d)
        print("Number of documents retrieved:")
        print(len(d))
        print("Names of documents are:")
        for f in d:
            print(file_names[f])
    else:
        print("phrase not present!")
else:
        search_phrase_query(query)
```

```
Please enter your queryOrganization: Utah State University
Number of documents retrieved:
1
Names of documents are:
sw_err.txt
```

This is the sample query on which code is tested and this is the format in which output will be shown , number as well as names of those docs will be printed.