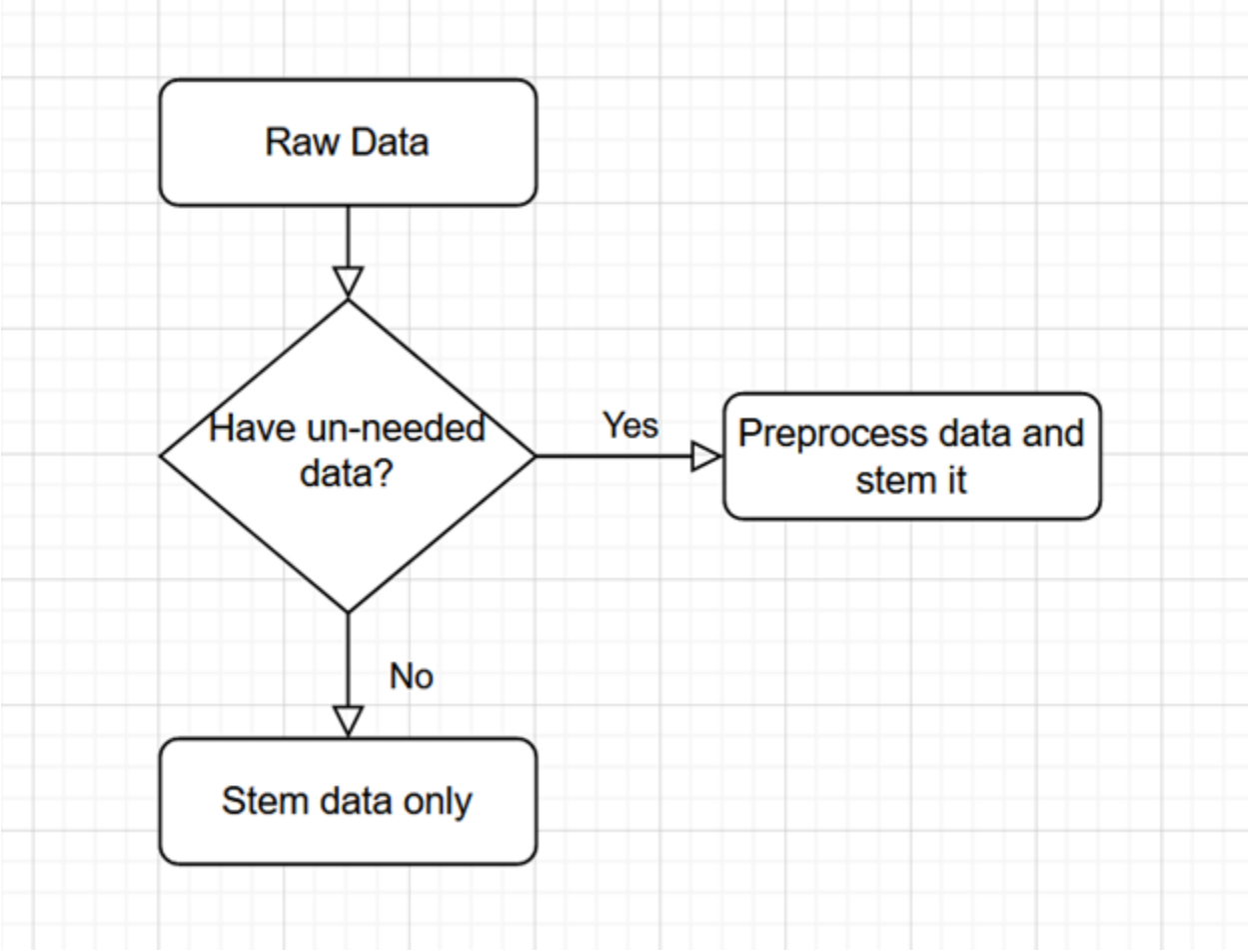
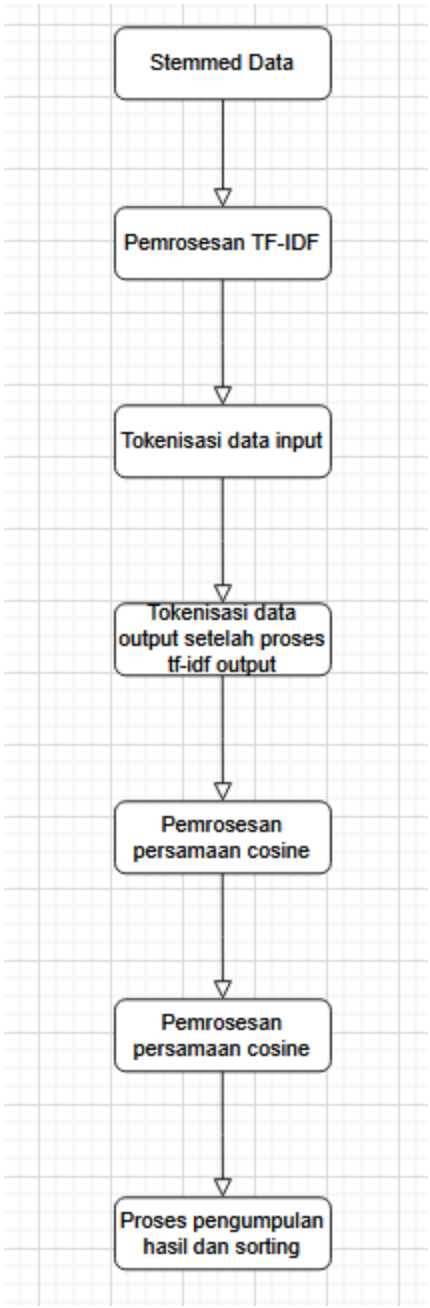
Laporan Eksekusi IR System

1872001 - Andrew Adrianus
1872007 - Axl Bintang
1872025 - Vardina Nava

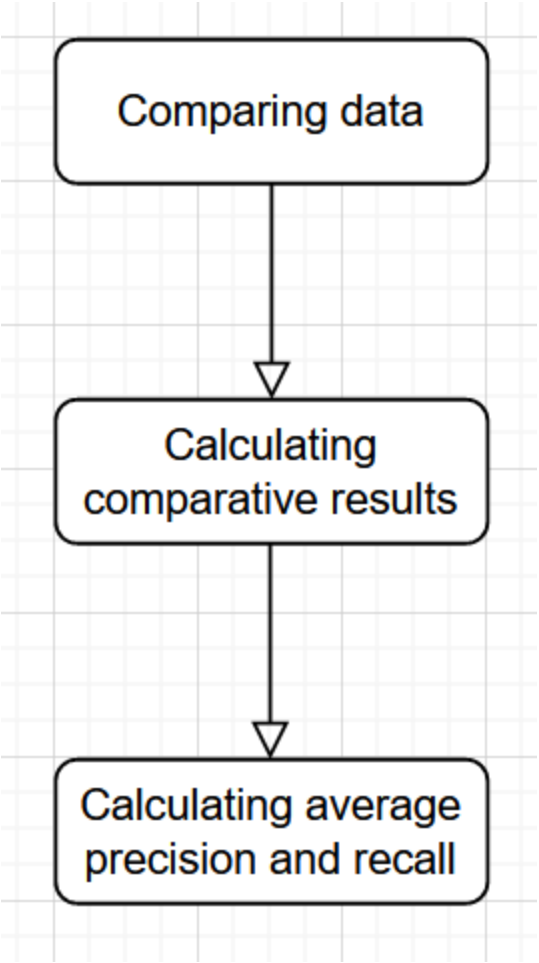
1.
Preprocess



Driver


Stemmed Data
Pemrosesan TF-IDF
Tokenisasi data input
Tokenisasi data output setelah proses tf-idf output
Pemrosesan persamaan cosine
Pemrosesan persamaan cosine
Proses pengumpulan hasil dan sorting

Out



2. Struktur data yang digunakan adalah penggabungan antara query dengan dictionary, query pada saat awal sebelum tf-idf dilakukan dan dictionary-query pada saat sesudah tf-idf.
Data Raw berbentuk txt. Data hasil preprocess berupa array, lalu dalam tahap tf idf berupa dictionary yang menampung representasi query & dokumen. Key dictionary adalah data kata yang sudah di preprocess dan valuenya adalah nilai tf idf nya. Hasil output tf idf berupa dict.

3. Precision dihitung berdasarkan count relevant/n, dimana n adalah jumlah keseluruhan kata yang ada pada query. Sementara Recall dihitung berdasarkan count recall / kata dalam list denominator.

4. Dokumen yang diberikan hanya ada queries dan relevance, sehingga kami mencoba memakai file dokumen buatan sendiri. Tetapi hasil array yang diberikan berupa array kosong dan precision 0.

```
Please enter the name of Queries file, you can also input the address of file:/content/Search-engine/Evaluation_data/queries.txt
Enter the path to file: /content/Search-engine/Evaluation_data/dokumen.txt
This is the output of query and relevant documents in descending order
[]
Please enter the name of relevance file, you can also input the address of file:/content/Search-engine/Evaluation_data/relevance.txt
FOR  10  MOST RELEVANT DOCUMENTS
Average precision 0.0
Average recall 0.0
FOR  50  MOST RELEVANT DOCUMENTS
Average precision 0.0
Average recall 0.0
FOR  100  MOST RELEVANT DOCUMENTS
Average precision 0.0
Average recall 0.0
FOR  500  MOST RELEVANT DOCUMENTS
Average precision 0.0
Average recall 0.0
```

5. Karena tidak ada file dokumen yang sesuai maka setidaknya pada nomor 5 ini kami akan menjelaskan step by step alur kerja sistem.
Bagian-bagian kode yang dipisah enter akan kami sebut sebagai bagian.
Bagian Driver :

```
file_name=input("Please enter the name of Queries file, you can also input the address of file:")
query= open(file_name,"r")          #Following code snippit is to tokenize query data and calculate tfidf values for it.
query_line= query.readline()
query_data=dict()
count=1

while(query_line):
    k=preprocess(query_line)
    query_data.update({count: k})
    query_line=query.readline()
    count+= 1

qindex=dict()                       # This is used to calculate tf for each word in a document
for key,value in query_data.items():
    for each in value:
        if each not in qindex:
            qindex.update({each:{key : 1}})
        else:
            if key not in qindex[each]:
                qindex[each].update({key : 1})
            else:
                qindex[each][key]=qindex[each][key]+1

qidf=dict()                         # This is used to calculate the IDF for each word in queries w.r.t. documents.
for key,value in qindex.items():
    id=math.log(10/(len(value)+1),10)
    qidf.update({key: id})

q_tfidf=dict()                      #Following is used to calculate the TFIDF value for rach word in each document(TF*IDF).
for key,value in qindex.items():
    for key1,value1 in value.items():
            if key1 not in q_tfidf:
                q_tfidf.update({key1:{key:qidf[key]*value1}})
            else:
                q_tfidf[key1].update({key:qidf[key]*value1})

qdoc_len=dict()                     #Following is used to calculate the document length for each query.
for key, value in q_tfidf.items():
```

Bagian pertama, program meminta input path file query, dalam contoh ini queries.txt.

```
Please enter the name of Queries file, you can also input the address of file:/content/Search-engine/Evaluation_data/queries.txt
Enter the path to file: /content/Search-engine/Evaluation_data/dokumen.txt
```

Setelah meminta input file, data dimasukkan ke dalam variabel dict().
Bagian selanjutnya, preprocess dilakukan terhadap variabel dict tersebut.
Bagian ketiga dan seterusnya, dilakukan penghitungan TF-IDF.

```
address= input("Enter the path to file: ")
folder = glob.glob(address + "/*")
data=dict()                         #Following is used to take data as input ans tokenize it.
for file_list in folder:
    file_open=open(file_list,'r')
    file_read=file_open.read()
    k=preprocess(file_read)
    x=file_list[-13:]
    data.update({x: k})

index=dict()                        #Following is used to calculate TF value for rach word in each document.
for key,value in data.items():
    for each in value:
        if each not in index:
            index.update({each:{key : 1}})
        else:
            if key not in index[each]:
                index[each].update({key : 1})
            else:
                index[each][key]=index[each][key]+1

idf=dict()                          #Here we calculate the IDF for rach document.
for key,value in index.items():
    id=math.log(1400/(len(value)+1),10)
    idf.update({key: id})

tfidf=dict()                        #Here we calculate the TFIDF for each word in each document.
for key,value in index.items():
    for key1,value1 in value.items():
            if key1 not in tfidf:
                tfidf.update({key1:{key:idf[key]*value1}})
            else:
                tfidf[key1].update({key:idf[key]*value1})

doc_len=dict()                      #This will calculate the leach of each document that helps in finding the cosine similarity.
for key, value in tfidf.items():
    s=0
    for key1, value1 in value.items():
        s= s + pow(value1,2)
```

Input kedua adalah input path file data.

```
Please enter the name of Queries file, you can also input the address of file:/content/Search-engine/Evaluation_data/queries.txt
Enter the path to file: /content/Search-engine/Evaluation_data/dokumen.txt
```

Bagian pertama file dibaca dan dimasukkan ke dalam dictionary untuk setiap file di dalam folder data, dilakukan preprocessing.
Setelah itu dilakukan perhitungan TF-IDF dan mencari persamaan kosinus.

```
relevant = dict()                   #Here we find out all the relevant documents to the specified queries.
for key1,value1 in query_data.items():
    inter=dict()
    for key,value in data.items():
        common=[]
        for each in value1:
            if each in value:
                common.append(each)
        inter.update({key:common})
    relevant.update({key1 : inter})

cosine=dict()                       #Cosine similarity is calculated for each query w.r.t. each document.
num=dict()
for key, value in relevant.items():
    for key1, value1 in value.items():
        xf=0.0
        den=0.0
        for each in value1:
            a=q_tfidf[key][each]
            b=tfidf[key1][each]
            xf=xf+(a*b)
        den=math.sqrt(doc_len[key1]*qdoc_len[key])
        final_cosine=xf/den
        if key not in num:
            num.update({key : {key1 : final_cosine}})
        else:
            num[key].update({key1: final_cosine})

sorted_val=dict()                   #Used to sort the documents in the descending order of their cosine similarity.
for key, value in num.items():
    sorted_x = sorted(value.items(), key=operator.itemgetter(1), reverse=True)
    sorted_val.update({key: sorted_x})
print("This is the output of query and relevant documents in descending order")
output_list=[]
for key, value in sorted_val.items():
    for i in range(0,len(value)):
        output_list.append([key,value[i][0]])
print(output_list)
```

Selanjutnya menampilkan array dari dokumen yang relevan dengan query.

```
This is the output of query and relevant documents in descending order
[]
```

```
file_name=input("Please enter the name of relevance file, you can also input the address of file:")
relevant= open(file_name,"r") # This snippet reads the "relevent.txt file"
r_line= relevant.readline()
r_data=dict()
while r_line:
    x = []
    x = r_line.split()              #We take input line for line and split the values of query and document.
    if int(x[0]) not in r_data:
        k=[int(x[1])]
        r_data.update({int(x[0]) : k})
    else:
        r_data[int(x[0])].append(int(x[1]))
    r_line= relevant.readline()

recall_denom=[]                     # This is to caluculate the number of relevant documents for each query.
for key,value in r_data.items():
    recall_denom.append(len(value))

output(r_data,sorted_val,recall_denom,10)     #Following give the answer to all the questions asked
output(r_data,sorted_val,recall_denom,50)
output(r_data,sorted_val,recall_denom,100)
output(r_data,sorted_val,recall_denom,500)
```

Input terakhir adalah path ke file relevances.txt
Data relevance juga dimasukkan ke dalam variabel bertipe dictionary. Setelah itu, setiap baris di relevance diperiksa untuk menemukan jumlah dokumen yang relevan dari setiap query.
Terakhir, function output yang berada di Out.py dipanggil untuk menampilkan urutan dokumen dari yang paling relevan.