Documents

1.txt

A paragraph is a series of sentences that are organized and coherent, and are all related to a single topic. Almost every piece of writing you do that is longer than a few sentences should be organized into paragraphs. This is because paragraphs show a reader where the subdivisions of an essay begin and end, and thus help the reader see the organization of the essay and grasp its main points.

2.txt

Paragraphs can contain many different kinds of information. A paragraph could contain a series of brief examples or a single long illustration of a general point. It might describe a place, character, or process; narrate a series of events; compare or contrast two or more things; classify items into categories; or describe causes and effects. Regardless of the kind of information they contain, all paragraphs share certain characteristics. One of the most important of these is a topic sentence.

3.txt

SCIENTISTS HAVE LEARNED TO SUPPLEMENT THE SENSE OF SIGHT IN NUMEROUS WAYS. In front of the tiny pupil of the eye they put, on Mount Palomar, a great monocle 200 inches in diameter, and with it see 2000 times farther into the depths of space. Or they look through a small pair of lenses arranged as a microscope into a drop of water or blood, and magnify by as much as 2000 diameters the living creatures there, many of which are among man's most dangerous enemies. Or, if we want to see distant happenings on earth, they use some of the previously wasted electromagnetic waves to carry television images which they re-create as light by whipping tiny crystals on a screen with electrons in a vacuum. Or they can bring happenings of long ago and far away as colored motion pictures, by arranging silver atoms and color-absorbing molecules to force light waves into the patterns of original reality.

Output

```
+-----------------+---------+---------+---------+
|                 |   1.txt |   2.txt |   3.txt |
+=================+=========+=========+=========+
| 200             |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| 2000            |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| a               |       1 |       1 |       1 |
+-----------------+---------+---------+---------+
| advises         |       0 |       1 |       0 |
+-----------------+---------+---------+---------+
| ago             |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| all             |       1 |       0 |       0 |
+-----------------+---------+---------+---------+
| almost          |       1 |       0 |       1 |
+-----------------+---------+---------+---------+
| among           |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| an              |       1 |       1 |       1 |
+-----------------+---------+---------+---------+
| and             |       1 |       1 |       1 |
+-----------------+---------+---------+---------+
| another         |       0 |       1 |       0 |
+-----------------+---------+---------+---------+
| are             |       1 |       0 |       1 |
+-----------------+---------+---------+---------+
| arranged        |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| arranging       |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| as              |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| at              |       0 |       1 |       0 |
+-----------------+---------+---------+---------+
| atoms           |       0 |       0 |       1 |
.                      .
.                      .
.                      .
.                      .
| beam            |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| because         |       1 |       0 |       0 |
+-----------------+---------+---------+---------+
| been            |       0 |       0 |       1 |
+-----------------+---------+---------+---------+
| before          |       0 |       1 |       0 |
+-----------------+---------+---------+---------+
| begin           |       1 |       0 |       0 |
+-----------------+---------+---------+---------+
| beginning       |       0 |       1 |       0 |
+-----------------+---------+---------+---------+
```

1.Aim: Create term-document incidence matrix and process Boolean queries.

Source Code:

```
import os
from tabulate import tabulate
os.chdir("C:/Users/rvr/Desktop/OMPR")
l=os.listdir()
d={}
totalwords=[]
for i in l:
   f=open(i,"r")
   q=f.read().lower()
   q=q.split()
   for x in q:
      if(not(x.isalnum())):
         p=list(x).copy()
         for t in x:
if(not t.isalnum()):
p.remove(t)
               flag=0
         if(flag==0):
            q[q.index(x)]=''.join(p)
   s=sorted(list(set(((q)))))
totalwords+=s
   d[i]=s
f.close()
totalwords=sorted(list(set(totalwords)))
w=[]
for x in totalwords:
   e=[]
e.append(x)
   for y in d:
if(x in d[y]):
e.append(1)
      else:
e.append(0)
w.append(e)
print(tabulate(w,headers=['Words']+l,tablefmt='grid'))
def finding(g):
   e=[]
if(g in totalwords):
      k=totalwords.index(g)
      for i in range(1,len(l)+1):
         if(w[k][i]==1):
e.append(l[i-1])
   return e
print("Searching with the help of Term Index Matrix!!!")
```

Output

1.
```
Searching with the help of Term Index Matrix!!!
Enter a word to search in available Collection:where
and whipping
No Related Documents found!!!
```

2.
```
Searching with the help of Term Index Matrix!!!
Enter a word to search in available Collection:photo or
inches
Related documents for given query are: 1.txt 3.txt
```

```
a=input("Enter a Boolean Query to search in available Collection:").lower()
a=a.split()
d=[]
result=[]
f=[]
s=[]
top=-1
temp=1
for x in range(0,len(a)):
   if(temp==0):
      temp=1
      continue
   if(a[x] in ['not']):
f.append(set(l)-set(finding(a[x+1])))
      temp=0
elif(a[x] not in ["and","or"]):
f.append(set(finding(a[x])))
elif(a[x]=='and'):
      top+=1
s.append('and')
elif(a[x]=='or'):
      while(top>-1 and s[top]=='and'):
         g=f[len(f)-1]
         h=f[len(f)-2]
f.remove(g)
f.remove(h)
f.append(g.intersection(h))
         top-=1
      top+=1
s.insert(top,'or')
while(top!=-1):
   g=f[len(f)-1]
   h=f[len(f)-2]
f.remove(g)
f.remove(h)
   if(s[top]=='and'):
f.append(g.intersection(h))
elif(s[top]=='or'):
f.append(g.union(h))
   top-=1
#print(f)
if(len(list(f[0]))==0):
print("No Related Documents found!!!");
else:
print("Related documents for given query are:",end=' ')
   for x in f:
      for y in sorted(list(x)):
print(y,end=' ')
```

Output

```
200 : 3.txt
2000 : 3.txt
a : 1.txt->2.txt->3.txt
advises : 2.txt
ago : 3.txt
all : 1.txt
almost : 1.txt->3.txt
among : 3.txt
an : 1.txt->2.txt->3.txt
and : 1.txt->2.txt->3.txt
another : 2.txt
are : 1.txt->3.txt
arranged : 3.txt
arranging : 3.txt
as : 3.txt
at : 2.txt
.
.
.
atoms : 3.txt
away : 3.txt
back : 3.txt
background : 2.txt
be : 1.txt->2.txt
beam : 3.txt
because : 1.txt
been : 3.txt
before : 2.txt
begin : 1.txt
beginning : 2.txt
best : 2.txt
blood : 3.txt
brief : 1.txt
bring : 3.txt
```

2. Aim: Build inverted index and process Boolean queries.

Source Code:

```
import os
os.chdir("C:/Users/rvr/Desktop/OMPR")
l=os.listdir()
d={}
totalwords=[]
for i in l:
   f=open(i,"r")
   q=f.read().lower()
   q=q.split()
   for x in q:
      if(not(x.isalnum())):
         p=list(x).copy()
         for t in x:
if(not t.isalnum()):
p.remove(t)
               flag=0
         if(flag==0):
            q[q.index(x)]=''.join(p)
   s=sorted(list(set(((q)))))
totalwords+=s
   d[i]=s
f.close()
totalwords=sorted(list(set(totalwords)))
w=[]
dic={}
for x in totalwords:
dic[x]=[]
   for y in d:
if(x in d[y]):
dic[x].append(y)
for i in dic:
   print(i,':','->'.join(dic[i]))

def finding(x):
   if x not in dic:
      return []
   return dic[x]
result=[]
print("Searching with the help of Inverted Index!!!")
a=input("Enter a BooleanQuery to search in available Collection:").lower()
a=a.split()
d=[]
result=[]
f=[]
```

Output

Output
1.
```
Searching with the help of Inverted Index!!!
Enter a word to search in available Collection:where
and whipping
No Related Documents found!!!
```

2.
```
Searching with the help of Inverted Index!!!
Enter a word to search in available Collection:photo or
inches
Related documents for given query are: 1.txt 3.txt
```

```
s=[]
top=-1
temp=1
for x in range(0,len(a)):
  if(temp==0):
    temp=1
    continue
  if(a[x] in ['not']):
f.append(set(l)-set(finding(a[x+1])))
    temp=0
elif(a[x] not in ["and","or"]):
f.append(set(finding(a[x])))
elif(a[x]=='and'):
    top+=1
s.append('and')
elif(a[x]=='or'):
    while(top>-1 and s[top]=='and'):
      g=f[len(f)-1]
      h=f[len(f)-2]
f.remove(g)
f.remove(h)
f.append(g.intersection(h))
      top-=1
    top+=1
s.insert(top,'or')
while(top!=-1):
  g=f[len(f)-1]
  h=f[len(f)-2]
f.remove(g)
f.remove(h)
  if(s[top]=='and'):
f.append(g.intersection(h))
elif(s[top]=='or'):
f.append(g.union(h))
  top-=1
if(len(list(f[0]))==0):
print("No Related Documents found!!!");
else:
print("Related documents for given query are:",end=' ')
  for x in f:
    for y in sorted(list(x)):
print(y,end=' ')
```

Output

```
===========Biwords in document: 1.txt =================
a paragraph
paragraph is
is a
a series
series of
of sentences
sentences that
that are
are organized
organized and
and coherent
coherent and
and are
are all
.
.
.
.
all related
related to
to a
a single
single topic
topic almost
almost every
every piece
piece of
of writing
writing you
you do
do that
that is
is longer
longer than
than a
a few
few sentences
sentences should
should be
```

3. Aim: Build biword index and process phrase queries.

Source Code:

```
import os
import numpy as np
os.chdir("C:/Users/omoni/Downloads/Text Files")
l=os.listdir()
d={}
for i in l:
   g=[]
   f=open(i,"r")
   s=(f.read().lower()).split()
   for x in s:
      if(not(x.isalnum())):
         p=list(x).copy()
         for t in x:
if(not t.isalnum()):
p.remove(t)
               flag=0
         if(flag==0):
            s[s.index(x)]=''.join(p)
   if(len(s)==1):
      g=s
   else:
      for x in range(1,len(s)):
g.append([s[x-1],s[x]])
   d[i]=g
f.close()
def finding(a,b):
   r=[]
   for x in range(0,len(d[b])):
      if(d[b][x]==a):
r.append(x)
   return set(r)
for x in d:
print("==========Biwords in document:",x,"===================")
   for y in range(0,len(d[x])):
print(' '.join(d[x][y]))
q=input("Enter Phrase Query to find in available documents:").split()
for x in q:
      if(not(x.isalnum())):
         p=list(x).copy()
         for t in x:
if(not t.isalnum()):
p.remove(t)
               flag=0
         if(flag==0):
```

Output

1.
```
Enter Phrase Query to find in available
documents:organized into paragraphs
[['organized', 'into'], ['into', 'paragraphs']]
Given Phrase Query is in document/documents: 1.txt
```

2.
```
Enter Phrase Query to find in available documents:read
where essay
[['read', 'where'], ['where', 'essay']]
Given Phrase not present in availabe documents!!!
```

```
            q[q.index(x)]=''.join(p)
w=[]
if(len(q)==1):
w.append(q)
else:
   for x in range(1,len(q)):
w.append([q[x-1],q[x]])
print(w)
result=[]
if(len(q)!=1):
   for x in l:
      for y in range(len(w)):
         if(y==0):
            r=finding(w[y],x)
         else:
            if(len(list(r))==0 and w[y] not in d[x]):
               break
            t=finding(w[y],x)
            r=t.intersection(set(np.array(list(r))+1))
      if(len(list(r))!=0 and y==len(w)-1):
result.append(x)
   if(len(result)!=0):
print("Given Phrase Query is in document/documents:",' '.join(result))
   else:
print("Given Phrase not present in availabe documents!!!")
else:
   flag=0
   for x in d:
      h=np.array(d[x]).flatten()
      if(w[0][0] in h):
print("Given Phrase Query is in document:",x)
         flag=1
   if(flag==0):
print("Given Phrase not present in availabe documents!!!")
```

Output

```
Resultant Positional Indexes are:
< a : 2
1  :  [1, 4, 18, 32, 45, 80, 84, 90, 95, 101, 107, 148]
3  :  [27, 50, 57, 60, 120, 125, 169, 184, 202]
>

< all : 1
1  :  [15, 135]
>

< almost : 2
1  :  [21]
3  :  [207]
>

< an : 2
1  :  [51]
3  :  [176]
>
>
.
.
< vacuum : 1
3  :  [126]
>
< we : 1
3  :  [89, 161, 198]
>
< which : 1
3  :  [80, 110]
>

< whipping : 1
3  :  [116]
>

< with : 1
3  :  [35, 122]
>

< x : 1
3  :  [189]
>
< yet : 1
3  :  [213]
>
```

4. Aim: Build positional index and process phrase queries.

Source Code:

```
import os
import numpy as np
os.chdir("C:/Users/rvr/Desktop/OMPR")
l=os.listdir()
result={}
for i in range(len(l)):
   f=open(l[i],'r')
   s=(f.read().lower()).split()
   for x in s:
      if(not(x.isalnum())):
         p=list(x).copy()
         for t in x:
if(not t.isalnum()):
p.remove(t)
               flag=0
         if(flag==0):
            s[s.index(x)]=''.join(p)
   for x in sorted(list(set(s))):
      q={}
      o=[]
      for y in range(0,len(s)):
         if(s[y]==x):
            o+=[y+1]
      q[i+1]=o
      if x not in result:
         result[x]=q
      else:
         result[x].update(q)
f.close()
print("Resultant Positional Indexes are:")
for x in result:
   print("<",x,":",len(result[x]))
   q=result[x]
   for i in q:
print(i," : ",q[i])
   print(">\n")
q=input("Enter the Phrase Query to search in available Collections:").split()
for x in s:
      if(not(x.isalnum())):
         p=list(x).copy()
         for t in x:
if(not t.isalnum()):
p.remove(t)
              flag=0
```

Output

1.
```
Enter Phrase Query to find in available
documents:organized into paragraphs
Given Phrase Query is in document/documents: 1.txt
```

```
2.
Enter Phrase Query to find in available documents:read
\4 essay
Given Phrase Query is in document/documents: 1.txt
2.txt

3.
Enter Phrase Query to find in available documents:read
\2 where
Given Phrase not present in availabe documents!!!
```

```
        if(flag==0):
           s[s.index(x)]=''.join(p)
w=set([int(i+1) for i in range(len(l))])
flag=0
e=set()
for x in w:
   e=set()
   d=set()
   for i in q:
if(i in result.keys()):
if(x in result[i].keys()):
            d=np.array(result[i][x])
            if(q.index(i)==0):
               d=d+1
               e=set(d)
            else:
               e=e.intersection(set(d))
               e=set(np.array(list(e))+1)
         else:
            e=set()
            d=set()
      else:
             print("Given Phrase not present in availabe documents!!!")
             quit()
if(len(list(e))!=0):
             print("Given Phrase Query is in document:",x)
             flag=1
if(flag==0):
             print("Given Phrase not present in availabe documents!!!")
```

Output

```
{'$a': ['a', 'a', 'are', 'and', 'and', 'are', 'all', 'a', 'almost',
'a', 'a', 'an', 'and', 'and', 'and', 'a', 'a', 'a', 'a', 'a', 'a',
'and', 'all', 'a', 'args[])', 'a', 'and', 'a', 'arranged', 'as', 'a',
'a', 'and', 'as', 'as', 'are', 'among', 'as', 'a', 'a', 'ago', 'and',
'away', 'as', 'arranging', 'atoms', 'and', 'a', 'an', 'a', 'and', 'a',
'almost'],
'a$': ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a'],
'$p': ['paragraph', 'piece', 'paragraphs.', 'paragraphs', 'points.',
'paragraphs',    'paragraph',    'point.',    'place,',    'process;',
'paragraphs',   'public',   'pupil',   'put,',   'palomar,',   'pair',
'previously', 'pictures,', 'patterns', 'penetrating', 'photograph.'],
'pa':   ['paragraph',   'paragraphs.',   'paragraphs',   'paragraphs',
'paragraph', 'compare', 'paragraphs', 'palomar,', 'space.', 'pair',
'patterns'],
 'ar':  ['paragraph',  'are',  'are',  'paragraphs.',  'paragraphs',
'paragraphs',   'paragraph',   'character,',   'narrate',   'compare',
'regardless', 'paragraphs', 'share', 'characteristics.', 'args[])',
'learned',  'palomar,',  'farther',  'arranged',  'are',  'earth,',
'carry',  'far',  'arranging'],  'ra':  ['paragraph',  'paragraph',
'paragraphs.', 'paragraphs.', 'paragraphs', 'paragraphs', 'grasp',
'paragraphs', 'paragraphs', 'paragraph', 'paragraph', 'illustration',
'general',  'character,',  'narrate',  'contrast',  'paragraphs',
'paragraphs',    'characteristics.',    'arranged',    'arranging',
'penetrating',   'rays,',   'photograph.',   'radiation'],   'ag':
['paragraph', 'paragraphs.', 'paragraphs', 'paragraphs', 'paragraph',
'paragraphs', 'magnify', 'electromagnetic', 'images', 'ago', 'images',
'electromagnetic'], 'gr': ['paragraph', 'paragraphs.', 'paragraphs',
'grasp',   'paragraphs',   'paragraph',   'paragraphs',   'great',
'photograph.'],  'ap':  ['paragraph',  'paragraphs.',  'paragraphs',
'paragraphs', 'paragraph', 'paragraphs', 'happenings', 'happenings',
'photograph.'],  'ph':  ['paragraph',  'paragraphs.',  'paragraphs',
'paragraphs',     'paragraph',     'paragraphs',     'photograph.',
'photograph.'], 'h$': ['paragraph', 'paragraph', 'with', 'through',
'much', 'which', 'which', 'with'], '$i': ['is', 'is', 'into', 'is',
'its',  'information.',  'illustration',  'it',  'items',  'into',
'information', 'important', 'is', 'in', 'in', 'inches', 'in', 'it',
'into', 'into', 'if', 'images', 'in', 'into', 'if', 'into', 'injured',
'information', 'it', 'into', 'images', 'in'], 'is': ['is', 'is',
'this', 'is', 'subdivisions', 'characteristics.', 'is', 'scientists',
'distant',  'television',  'discovered'],  's$':  ['is',  'series',
'sentences',  'is',  'sentences',  'this',  'is',  'paragraphs',
'subdivisions',  'thus',  'its',  'paragraphs',  'kinds',  'series',
'examples', 'series', 'items', 'causes', 'regardless', 'paragraphs',
'is', 'scientists', 'numerous', 'inches', 'times', 'depths', 'lenses',
'as', 'as', 'as', 'diameters', 'creatures', 'manâ€™s', 'dangerous',
'happenings',  'waves',  'images',  'as',  'crystals',  'electrons',
'happenings',  'as',  'atoms',  'molecules',  'waves',  'patterns',
'images', 'thus', 'has'], '$s': ['series', 'sentences', 'single',
'sentences',  'should',  'show',  'subdivisions',  'see',  'series',
'single', 'series', 'share', 'sentence.', 'static',
]}
```

5. Aim: Build bi-gram index and process wildcard queries.

Source Code:

```
def bigram(s):
   d=[]
   for i in range(len(s)-1):
      d+=[s[i]+s[i+1]]
   return d
def intersect(g,bigrams):
   r=set()
   p=1
   for x in g:
      if(p==1):
         r=set(bigrams.get(x,[]))
         p=0
      else:
         r=r.intersection(bigrams.get(x,[]))
   return r
import os
os.chdir("C:/Users/exam2/Desktop/OMPR")
l=os.listdir()
bigrams={}
totalwords=[]
pr=0
for x in l:
   f=open(x,"r")
   p=(f.read().lower()).split()
totalwords+=p
   s=list(map(lambda x:'$'+x+'$',p))
   for i in range(len(s)):
      for j in range(len(s[i])-1):
         if(s[i][j]+s[i][j+1] in bigrams.keys()):
            bigrams[s[i][j]+s[i][j+1]]+=[p[i]]
         else:
            bigrams[s[i][j]+s[i][j+1]]=[]
            bigrams[s[i][j]+s[i][j+1]]+=[p[i]]
f.close()
#print(bigrams)
totalwords=set(totalwords)
result=set()
single=[]
q=(input("Enter Query:").lower())
if('*' in q):
   if(q.count('*')==1):
      if(q.index('*')==0):      #*lo
         g=bigram(q[1:]+'$')
         result=intersect(g,bigrams)
```

```
elif(q.index('*')==len(q)-1):  #he*
          g=bigram('$'+q[:-1])
          result=intersect(g,bigrams)
       else:
          g=bigram('$'+q[:q.index('*')]) #h*o
          result=intersect(g,bigrams)
          g=bigram(q[q.index('*')+1:]+'$')
          result=result.intersection(intersect(g,bigrams))
    else:
       if(q[0]!='*' and q[len(q)-1]!='*'):   #h*l*o
          q='$'+q+'$'                  #h*el*o
          q=q.split('*')
          for x in q:
             if(len(x)!=1):
                g=bigram(x)
                if(q.index(x)==0):
                   result=intersect(g,bigrams)
                else:
                   result=result.intersection(intersect(g,bigrams))
             else:
single.append(x)
elif(q[0]=='*' and q[len(q)-1]=='*'):
          w=q[1:len(q)-1]
if('*' not in w):
             if(len(w)==1):
                for x in totalwords:
if(w in x):
print(x,end=' ')
                      pr=1
             else:
                g=bigram(w)
                result=intersect(g,bigrams)
                print(result)
                pr=1
          else:
             w=w.split('*')
             for x in totalwords:
                flag=0
                for y in w:
if(y in x):
                      flag=1
                   else:
                      flag=0
                      break
                if(flag==1):
print(x,end=' ')
                   pr=1
       else:
          if(q[0]=='*'):
```

Output

1.
```
Enter Query:p*g*h
Resultant Set is: paragraph
```

2.
```
Enter Query:wh*e
Resultant Set is: Where
```

```
            w=q[1:]
         else:
            w=q[:len(q)-1]
         w=w.split('*')
         for x in totalwords:
            flag=0
            for y in w:
if(y in x):
                  flag=1
               else:
                  flag=0
                  break
            if(flag==1):
print(x,end=' ')
               pr=1
flag=0
for x in result:
   flag=0
   for y in single:
      if y not in x:
         flag=1
         break
   if(flag!=1):
print('Resultant Set is:',x)
      pr=1
if(pr!=1):
print("Given Query is Not Valid!!!")
```

6. Aim: Implement spelling correction technique in information retrieval.

Source Code:

```
def editDistance(a, b):
   m, n = len(a), len(b)
   insertion,deletion,replace = 0,0,0
   ed = []
   for i in range(m + 1):
      row = []
      for j in range(n + 1):
         row.append(0)
      ed.append(row)
   for i in range(m + 1):
      ed[i][0] = i
   for j in range(n + 1):
      ed[0][j] = j
   for i in range(1, m + 1):
      for j in range(1, n + 1):
         if a[i - 1] == b[j - 1]:
               ed[i][j] = ed[i - 1][j - 1] # same character
         else:
            if((ed[i - 1][j] < ed[i][j - 1]) and (ed[i - 1][j] < ed[i - 1][j - 1])):
                cost = ed[i - 1][j]
            elif((ed[i][j - 1] < ed[i - 1][j]) and (ed[i][j - 1] < ed[i - 1][j - 1])):
                cost = ed[i][j - 1]
            else:
                cost = ed[i - 1][j - 1]
            ed[i][j] = 1 + cost
   print(ed)
   while m > 0 and n > 0:
      if ed[m][n] == ed[m-1][n-1]:
         m -= 1
         n -= 1
      elif ed[m][n] == ed[m-1][n] + 1:
         deletion += 1
         m -= 1
      elif ed[m][n] == ed[m][n-1] + 1:
         insertion += 1
         n -= 1
```

Output

1.
```
Enter term 1: beauty
Enter term 2: pretty
[[0, 1, 2, 3, 4, 5, 6], [1, 1, 2, 3, 4, 5, 6], [2, 2,
2, 2, 3, 4, 5], [3, 3, 3, 3, 3, 4, 5], [4, 4, 4, 4, 4,
4, 5], [5, 5, 5, 5, 4, 4, 5], [6, 6, 6, 6, 5, 5, 4]]
No. of insertions:  1
No. of deletions:  1
No. of replacements:  2
Edit Distance: 4
```

```
2.
Enter term 1: fast
Enter term 2: cats
[[0, 1, 2, 3, 4], [1, 1, 2, 3, 4], [2, 2, 1, 2, 3], [3,
3, 2, 2, 2], [4, 4, 3, 2, 3]]
No. of insertions:  1
No. of deletions:  1
No. of replacements:  1
Edit Distance: 3
```

```
        else:
            replace += 1
            m -= 1
            n -= 1
    while m > 0:
        deletion += 1
        m -= 1
    while n > 0:
        insertion += 1
        n -= 1
    print('No. of insertions: ',insertion)
    print('No. of deletions: ',deletion)
    print('No. of replacements: ',replace)
    return ed[len(a)][len(b)]

term1 = input("Enter term 1: ")
term2 = input("Enter term 2: ")
print("Edit Distance:", editDistance(term1, term2))
```