CercaStringa

Obiettivo

Il nostro programma ha 3 **input**:

- una directory di partenza in cui cercare
- una **parola** da cercare
- una **directory di output**

L'obiettivo è quello di trovare tutti i file che hanno a che fare con la **parola** inserita, nel pezzo di filesystem definito dall'utente. Tutti i file trovati saranno copiati nella **directory di output**.

Partiamo dal seguente codice

```
import os

#IMMISSIONE DEI PARAMETRI
sRoot = input("Inserisci la root directory: ")
sStringaDaCercare = input("Inserisci la stringa da cercare: ")
sOutDir = input("Inserisci la dir di output: ")

#NAVIGA NEL FILE SYSTEM
for root, dirs, files in os.walk(sRoot):
    sToPrint = "Dir corrente {0} contenente {1} subdir e {2}
     files".format(root, len(dirs), len(files))
    print(sToPrint);
```

Step successivo

Per ogni sottodirectory trovata andiamo a scorrere tutti i file e richiamiamo una funzione che verifica se il file contiene la parola cercata. Alla fine stampiamo il numero di file trovati.

```
def CercaStringaInFileName(sFile,sString):
    return False
```

```
for filename in files:
        iRet = CercaStringaInFileName(filename,sStringaDaCercare)
        if(iRet == True):
            print("Trovato file: ",filename);
            iNumFileTrovati = iNumFileTrovati + 1;
```

Step successivo

Arricchiamo la funzione CercaStringaInFile mettendoci la ricerca nella parte che riguarda il nome:

```
def CercaStringaInFilename(sFilename,sStringToSearch):
    sFilename1 = sFilename.lower();
    sStringToSearch1 = sStringToSearch.lower();
    print("Cerco {0} in {1} ".format(sFilename1,sStringToSearch1));
    iRet = sFilename1.find(sStringToSearch1);
    if(iRet>-1):
        print("Trovato");
        return True;
    return False;
```

Step successivo

Arricchiamo la funzione mettendoci la ricerca nella parte che riguarda il file letto come file binario, cioè byte a byte. Ovviamente, se dobbiamo accedere al file ci servirà il path completo del file stesso e non solo il filename come fatto finora.

```
for filename in files:
    #print(filename);
    iRet = CercaStringaInFileName(filename,sStringaDaCercare)
    if(iRet == True):
        print("Trovato file: ",filename);
        iNumFileTrovati = iNumFileTrovati + 1;
    else:
        pathCompleto = os.path.join(root,filename)
        iRet = CercaStringaInFileContent(pathCompleto ,sStringaDaCercare)
```

Step successivo

Creiamo quindi la funzione CercaStringaInFileContent dove facciamo una lettura del file byte a byte e per riga.

```
import mmap

def CercaStringaInFileContent(sFile,sString):
    sString = sString.lower();
    iLen = len(sString);
    #leggiamo il filename
    try:
        with open(sFile) as f:
            s = mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_READ)
            sAppo = s.readline()
            while len(sAppo)> 0 :
                sAppo = sAppo.lower();
                if(sAppo.find(sString.encode())!=-1):
                    return True
                else:
                    sAppo = s.readline()
    except:
        return False
```

Step successivo

Arricchiamo la funzione CercaStringaInFileContent mettendoci la ricerca nella parte che riguarda il file letto come file PDF. Ovviamente se le fasi precedenti di ricerca hanno dato esito negativo.

Come primo step dobbiamo riconoscere che un file è un file PDF. Adoperiamo un criterio debole ma semplice e lavoriamo semplicemente sull'estensione del file.

```
sOutFileName,sOutFileExt = os.path.splitext(sFile)
if(sOutFileExt.lower()==".pdf"):
    #print("Riconosciuto file pdf " + sFile)
    return CercaInFilePdf(sFile,sString)
```

Step successivo

Dobbiamo scrivere la procedura CercaInFilePDF. Dobbiamo cercare una libreria che interpreta i file PDF e trasforma in testo il contenuto.

pip3 install PyPDF2

```
import PyPDF2

def CercaInFilePdf(sFile,sString):
    object = PyPDF2.PdfFileReader(sFile)
    numPages = object.getNumPages()

    for i in range(0, numPages):
        pageObj = object.getPage(i)
        text = pageObj.extractText()
        text = text.lower()
        if(text.find(sString)!=-1):
            return True
    return False
```

Step successivo

Gli stessi step fatti per il PDF li possiamo fare per altri tipi di file. Per esempio potremmo considerare i file DOC.

Come primo step dobbiamo riconoscere che un file è un file DOC. Anche qui lavoriamo semplicemente sull'estensione del file. Ci ricordiamo però che un file doc può avere estensione doc oppure docx.

```
sOutFileName,sOutFileExt = os.path.splitext(sFile)
if((sOutFileExt.lower()==".doc") or (sOutFileExt.lower()==".docx")):
    #print("Riconosciuto file doc " + sFile)
    return CercaInFileDoc(sFile,sString)
```

Step successivo

Dobbiamo scrivere la procedura CercaInFileDOC. Dobbiamo cercare una libreria che interpreta i file DOC e trasforma in testo il contenuto.

```
import textract

def CercaInFileDoc(sFile,sString):
    text = textract.process(sFile)
    text = text.lower()
    if(text.find(sString.encode())!=-1):
                    return True
    return False
```

Step successivo

Gli stessi step fatti per il PDF e per il DOC potremmo farli per altri formati. Non li facciamo perché i passi sono sempre gli stessi:

- Cercare una libreria in python che interpreta il formato e lo restituisce come testo.
- Linkare la libreria al nostro programma.

Quello che faremo è una cosa diversa. Proveremo a gestire un formato particolare, quello delle immagini. Prima però terminiamo la struttura del nostro programma.

Step successivo

I file che riconosciamo li vogliamo copiare in una directory di output.

```
def SalvaFile(sFilePath,sFileName,sOutDir):
    os.makedirs(sOutDir, exist_ok=True)
    sOutFile = sOutDir + "/" + sFileName
    print("Devo copiare {0} in {1}".format(sFilePath,sOutFile)
    #shutil.copyfile(sFilePath,sOutFile)
```

Come si può notare la funzione di copia è commentata. C'è un motivo. La procedura cosi come messa ha due problemi. Quali sono?

Step successivo

La procedura della slide precedente, come detto, ha due problemi:

1) Due file con lo stesso nome ma presenti in due cartelle diverse verrebbero a sovrascriversi. Sopravviverebbe senza motivo solo l'ultimo trovato, in base alla sua posizione nel filesystem. E questo non va bene.
2) Appena troviamo il primo file, creiamo (se non esiste già) la directory di output. Questo potrebbe provocare il fatto che la ricerca si estende anche alla directory di output stesso. Generando un circuito potenzialmente infinito.

Che soluzione trovereste per questi due problemi?

Primo problema: nomi ripetuti

Due file con lo stesso nome ma presenti in due cartelle diverse verrebbero a sovrascriversi. Sopravviverebbe senza motivo solo l'ultimo trovato, in base alla sua posizione nel filesystem. E questo non va bene.

In questo caso l'idea è di riportare nel nome del file in output tutto il percorso. Perché se due file hanno nomi uguali sicuramente il percorso è univoco. L'idea è che se io cerco "cane" e trovo il file nella directory

C:\documenti\animali\cane_pastore.pdf

C:\documenti\cortile\cane_pastore.pdf

Nella directory di output indicata dall'utente salvo il file cane_pastore.pdf ma rinominandolo in

c_documenti_animali_cane__pastore.pdf

Primo problema: nomi ripetuti

```
sFilePathNew = sFilePath.replace(“_”,”__”)
sFilePathNew = sFilePathNew.replace("\\","_")
sFilePathNew = sFilePathNew.replace("/","_")
sOutFile = sOutSubDir + "/" + sFilePathNew
```

Secondo problema: ricerca ricorsiva

Appena troviamo il primo file, creiamo (se non esiste già) la directory di output. Questo potrebbe provocare il fatto che la ricerca si estende anche alla directory di output stesso. Generando un circuito potenzialmente infinito.

Per ovviare a questo problema quando abbiamo un file in cui cercare andiamo prima a verificare che lo stesso non si trovi in una sottocartella di quella indicata dall'utente.

Qui c'è un'accortezza: il confronto deve essere fatto con i due path assocluti altrimenti potrei avere match non desiderati. Il path relativo indicato dall'utente potrebbe essere presente in più path assoluti.

Secondo problema: ricerca ricorsiva

```
def in_directory(dir1, dir2):
    #make both absolute
    directory1 = os.path.abspath(dir1).lower()
    directory2 = os.path.abspath(dir2).lower()
    print("Dir1: ",directory1)
    print("Dir2: ",directory2)
    if(directory1.find(directory2)!=-1):
        return True
    else:
        return False
```

C’è un terzo problema

C’è un terzo problema: se faccio due ricerche successive, con la stessa directory di output, cosa che può essere normale, faccio confusione tra I risultati della prima ricerca ed I risultati della seconda.

Per ovviare al problema ogni ricerca è salvata in una directory, che prende il nome della data specificata fino ai secondi, in cui la ricerca è stata effettuata.

```
now = datetime.now()
 dt_string = now.strftime("%Y_%m_%d_%H_%M_%S")
 sOutSubDir = sOutDir + "/" + dt_string
 os.makedirs(sOutSubDir, exist_ok=True)
```

Piccoli ritocchi

Alla fine del programma stampiamo a schermo il numero di file che soddisfano la stringa cercata. Potrebbe essere interessante stampare il numero in relazione ai file cercati. Es. “trovati 4 file su 100 file presenti” . Potremmo segnalare in modo distinto il fatto che la **directory indicata** è vuota.

Inoltre potremmo voler segnalare all’utente il caso in cui ci indica una directory non esistente:

os.path.isdir(path)

che torna False se la directory non esiste.