Exercise 1

For this exercise it was used the output of the Exercise 6 from the first homework, that contains all the jobs. From that file a pre-processing has been done. This means that characters like $\sim$,$*$ or { have been removed. Also stopwords from the italian language have been removed, like punctuation signs. The entire pre-processing resulted in the *result.txt* file, that contains in each line a different job.
After the pre-processing, there were constructed two different indexes. The first one is called *EuclideanDistancesIndex.txt* and contains pairs (jobID, euclidean distance), comma separated. The euclidean distance of a job is the square root of the sum over all squares of each number of occurrences of each word contained in the job. The second index is the inverted index. The inverted index file, *InvertedIndex.txt* contains, in each line, the word, the document frequency of that word (that is the number of documents containing the word), and all the pairs (jobID, number of occurrences of the word in the job). The two values inside the pairs are comma separated, meanwhile : is used to separate all the fields. The following line is an example:
school:4:1,3:2,1:5,1:6,2
means that the word school is contained in 4 jobs, in the job with id=1 it is contained 3 times, in the job with id=2 just once, and so on.
The cosine similarity between a query and a job was computed in the following way:

for each term t of the query q
    compute weight $w_{t,q}$ *of t in q*
        for each job in the postings list of t
            compute weight $w_{t,job}$ *of t in job*
                score[job] $+= w_{t,job} \cdot w_{t,q}$
for each job
    scores[job] /= distance[job]
return the top 10 scores

The formula used of the weight is the *tf idf* formula, that is the product between the term frequency and the inverse document frequency of some term t in a job j.

How to run
The name of the file is *hw2ex.py*, and it can be run from terminal with the *python hw2ex.py* with 2.7 version of Python. In one run it will create all the files needed for computing the queries, that are *result.txt*, *InvertedIndex.txt*, *WordDocIDpairsSorted.txt* and *EuclideanDistancesIndex.txt*.
Next, since all the needed indexes were created, one can comment from the *main* function all the calls but *computeProximityQuery()*, in order just to compute the proximity queries, without re-creating each time the indexes.

Examples of queries
Since the editor counts the lines from 1, to the output index of the terminal 1 must be added.
Query 1: *programmazione orientata agli oggetti*

```
The 10 most similar job announcements are
document index: 369 with score: 1.77292727269
document index: 1251 with score: 1.027235234
document index: 3437 with score: 0.994848112158
document index: 3755 with score: 0.834657746044
document index: 2991 with score: 0.831872611535
document index: 850 with score: 0.815757384496
document index: 102 with score: 0.812770171817
document index: 3687 with score: 0.778634792032
document index: 982 with score: 0.694065789789
document index: 2594 with score: 0.676416205068
```

```
370   sviluppatore programmazione oggetti divisione ict openjobmetis
      spa specializzata reclutamento selezione gestione risorse umane
      ambito informatico ricerca propria azienda cliente sviluppatore
      programmazione oggetti  ltmazienda ricercando programmatore
      oggetti inserire proprio team sviluppo implementare nuove
      funzioni software aziendale uso interno richiesti almeno anni
      esperienza dimostrabile sviluppo applicazioni conoscenza
      delltmintero sviluppo prodotto progettazione testing rilascio
      manutenzione istruzione richiesta  diploma laurea informatica
      materie scientifiche  conoscenza lingua inglese livello parlato
      scritto competenze richieste  comprovata esperienza sviluppo
      metodologia programmazione oggetti  buona conoscenza seguenti
      linguaggi programmazione java .net  ms sql server  preventivare
      tempi metodi sviluppo rilascio software  design patterns mvp
      conoscenza processi aziendali produzione gestione magazzino
      acquisti  principi programmazione solid completano profilo
      capacita analisi problem solving lavoro gruppo autonomia
      operativa sede lavoro provincia palermo ltminquadramento
      contrattuale proposto valutato base esperienze competenze
      maturate candidati selezionati verranno contattati
      telefonicamente fissare primo colloquio conoscitivo presso sedi
      lannuncio rivolto personale ambosesso riferimento lgs titolare
      openjobmetis agenzia lavoro sede legale via generale fara milano
      informa trattamento dati personali forniti avverra mediante
      strumenti manuali informatici telematici invitano candidati
      trasmettere esclusivamente dati necessari valutare proprio
      profilo professionale inviare dati sensibili dati trattati
      personale openjobmetis potranno essere comunicati utilizzatori
      interessati avvalersi attivita lavorativa dati diffusi conoscere
      modalita esercizio diritti conferiti avere ulteriori informazioni
      consulti privacy policy aut prot palermo
```

Query 2: *programmatore c++*

```
Insert the query: programmatore c++
The 10 most similar job announcements are
document index: 1865 with score: 1.58712283149
document index: 3486 with score: 1.25181814037
document index: 982 with score: 1.10405283269
document index: 3075 with score: 1.05553948241
document index: 1175 with score: 0.963825687762
document index: 3301 with score: 0.963089401931
document index: 2573 with score: 0.872387209309
document index: 380 with score: 0.79901641438
document index: 3941 with score: 0.721612695307
document index: 1741 with score: 0.675495828098
```

```
1866  php c++ mysql mariadb rabbitmq linux freebsd unix seo webagency
      ciao nerd partita iva vuoi crearti futuro gia entrate sostenerti
      cerchi altre entrate conosciamoci parliamo progetti tag php c++
      mysql mariadb rabbitmq linux freebsd unix seo webagency lavoro
      collaborazione react angular programmatore sviluppatore want you
      solo residenti provincia bergamo orio serio
```

Long query 3: *addetta copisteria fotocopie grafica ricerchiamo ragazza addetta fotocopie buona conoscenza adobe illustrator photoshop nonche creativita grafica zona lavoro roma tuscolana tuscolano don bosco cinecitta*

```
[MacBook-Pro-di-Ciprian:HW2 mychro94$ python hw2.py
Insert the query: addetta copisteria fotocopie grafica ricerchiamo ragazza addet
ta fotocopie buona conoscenza adobe illustrator photoshop nonche creativita graf
ica zona lavoro roma tuscolana tuscolano don bosco cinecitta
The 10 most similar job announcements are
document index: 2363 with score: 9.76545370227
document index: 959 with score: 6.73799268109
document index: 2454 with score: 3.02109792539
document index: 1610 with score: 2.68260066314
document index: 2504 with score: 1.93194702208
document index: 2336 with score: 1.8386061153
document index: 586 with score: 1.77035638523
document index: 1380 with score: 1.72604881656
document index: 104 with score: 1.6648667355
document index: 2149 with score: 1.54016324178
MacBook-Pro-di-Ciprian:HW2 mychro94$
```

```
2364  addetta copisteria fotocopie grafica ricerchiamo ragazza addetta
      fotocopie buona conoscenza adobe illustrator photoshop nonche
      creativita grafica zona lavoro roma tuscolana tuscolano don bosco
      cinecitta
```

```
960   programmatore app mobile societa ricerca sviluppo espansione
      cerca interessante progetto video streaming intelligenza
      artificiale sviluppatore piattaforme mobile requisiti necessari
      ottima conoscenza html5 css3 javascript ottima conoscenza
      sviluppo ambito android ios ottima conoscenza sistemi informativi
      database buona esperienza lavoro tecnologie rest ws soa prevede
      lassunzione tempo indeterminato full time zona tor vergata
      retribuzione sicuro interesse tuscolano don bosco cinecitta
```

Exercise 2

Part 1

The jobs documents contains each job on a line, so for the shingling part each line was read, shingles of length k were created, then each shingle was hashed with the *binascii.crc32* hash function, and added to a list. This list finally is the value of the job id in a dictionary. The hash function takes the shingles and transforms it into a four-bytes integer. This work is done by the *JobShingling(job, k)* function. During the reading of the lines, also another dictionary was creating, containing for each job id, a value that is just the set of words of that job. This will be useful for computing the Jaccard similarity on the jobs.

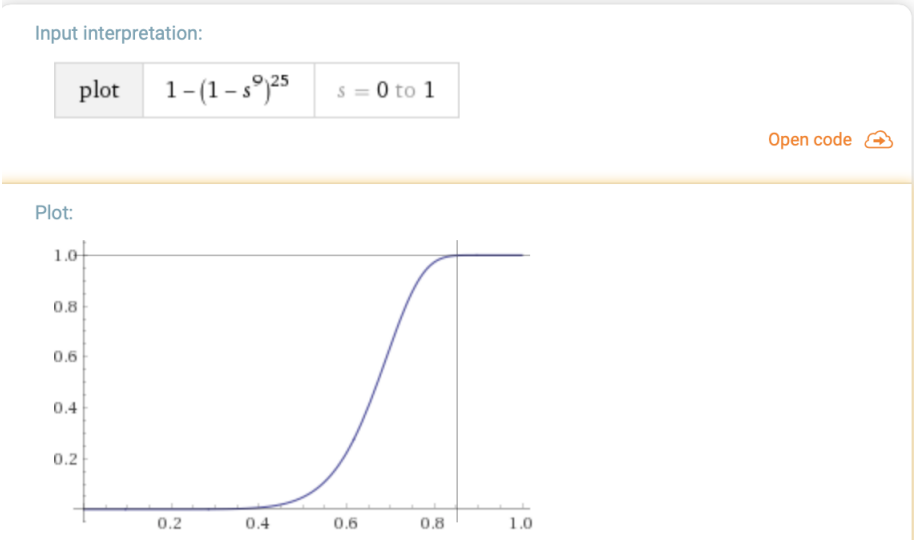
Part 2

Now we need to create, for each job, its signature. The number of random hashing function chosen, l, is 225. The hashing function chosen is
$h(x) = (ax + b) mod\ c$
Therefore, the two calls to the *pickRandomCoeffs(l)* function, there were generated l distinct values for a and l distinct values for b. Since a and b are integers between 0 and $2^{32}-1$, c will be the next prime number after 2^{32}. Then, the *computeSignatures(shinglesDict, coeffA, coeffB, l)* function is called, that returns a

dictionary containing, for each job id, its signature. The length of the signature will be the number of hashing functions used, therefore l. For each job id, a list long l is initialized, and for each of the l hashing functions, all the shingles of that job were hashed, and the minimum resulting number was chosen, and placed into the list at the index belonging to that hashing function. Since this work is done for each hashing function, and for each job, in the end we will have a dictionary with the job id as key, and value the list long l of minimum hashes, that represents the signature for that job. We will see that this function is very time-consuming, in fact computing all the signatures of over 4000 jobs announcements will take around 120 seconds.
Moreover, comparing all the possible pairs of signatures is even more time-consuming, in fact we see that it takes around 170 seconds.

```
MacBook-Pro-di-Ciprian:HW2 mychro94$ python hw2ex2.py
Calculating all the signatures took 117.25sec
Calculating all minwise hashing pairs took 167.89sec
MacBook-Pro-di-Ciprian:HW2 mychro94$
```

Part 3 Now, since the *part 2* takes so much time, we would like to enhance the process of finding similar jobs. Since l hashing function were used, that is 225, the number of bands b will be equal to 25, and the number of rows r will be 9. This guarantees a pretty good S-curve.



Of course, this is a tradeoff between the amount of hashing functions number, and the number of false positives that may arise.
For this part, the *LSHresults(signatures)* function was called. This will call the function that will return all the candidate pairs of jobs, that is *LSH(signatures, b, r)*. The latter will compute the hashing of the r elements of the signatures, for each signature, b times, that is the number of bands. The hashing of the r elements is just the module operation of the sum of the r integers in the signature. For the module, the next prime after 2^{32} is used. To keep trace of all the jobs that hashed to the same bucket, we keep a dictionary that has as key the result of the hash, and as value thelist of job ids that hashed to that bucket. This dictionary is the returned, only with the keys that have a list longer than 1. Once returned, for each pair of documents in the dictionary, their signatures

are fetched, and the similarity between them is computed. We will see that the whole process will just take around a second.

```
Calculating all LSH pairs took 1.05sec
The number of duplicate pairs found with LSH are 2302
```

This represents a significant improvement wrt the minwise hashing comparisons.

Task 1
The number of duplicates pairs found with LSH are 2392, number that can be observed in the image above.

Task 2
Here we want to find the near-duplicates, pairs with over 80% of similarity, using the Jaccard similarity.

```
The number of duplicate pairs found with Jaccard with similarity at least 80% are 1792
```

Task 3
The intersection between the LSH results and Jaccard results is a set of 1731 common pairs. This means that LSH found $\frac{1731}{1792} \cdot 100 = 96.5\%$ of the pairs with a similarity higher than 80%.

Task 4
The time required for the LSH, with pre-computed signatures, is around 1 second.
The time required for the Jaccard is around 250 seconds.

```
Calculating all LSH pairs took 1.05sec
Calculating all Jaccard Similarities took 247.55sec
```

How to run
From the terminal, run *python hw2ex2.py*, with Python 2.7.

Exercise 3

PySpark was used for implementing the exercise 2 in Apache Spark. I first did the pseudocode for Hadoop MapReduce framework, then implemented it with *PySpark*.
I can split the problem in two parts, one finding the near-duplicates pairs of job announcements according to the Jaccard similarity, and the other one will be finding the candidate pairs according to the LSH technique.

Jaccard MapReduce

```
map(jobID, words):
    foreach word in words:
    emit(word, (len(words), jobID))

reduce(word, list(tuples(length, jobID))):
    foreach pair of tuples k, v in the list:
        emit((k,v), 1)
```

map(k, v):
    emit(k, v)

reduce((job1, job2), list(ones)):
    emit((job1.id, job2.id), similarity $= \frac{len(list)}{job1.length + job2.length - len(list)})$

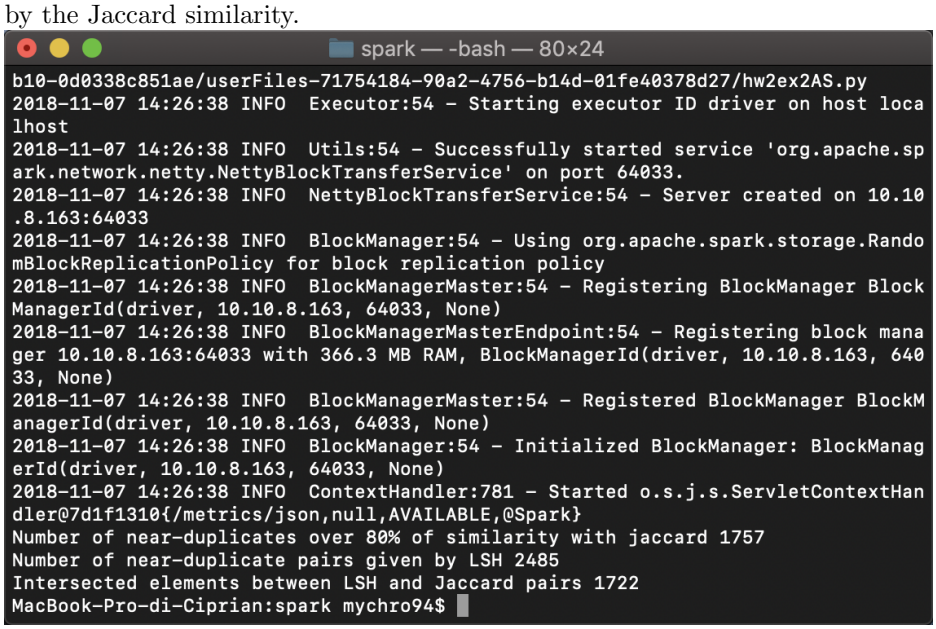
LSH MapReduce
map(jobID, words):
    shinglesSet.add(shingle(words))
    foreach s in shinglesSet:
        hash(s)
    signature = []
    foreach hashing function i:
        minhash = min(all h = $hash_i(s)$ *of s in shinglesSet)*
        signature[i] = minhash
    for i in range(0, bands):
        emit(h = hash(signature[ir:ir+r]), jobID)

reduce(hash, list(colliding Jobs)):
    foreach unique pair in list:
    emit(pair)

Once we have computed all the pairs, in both cases, we only need to intersect the two sets and see the pairs.

Example

It can be observed that the percentage of near-duplicate jobs found by the LSH technique covers 98% of the real pairs with more than 80% of similarity given by the Jaccard similarity.



How to run
From the terminal, once in the */urs/local/spark* location, give the command */bin/spark-submit hw2ex2AS.py*. This is for MacOS Mojave systems with 2.3.2 *spark* version. Also, the *hw2ex2AS.py* must be moved in the same directory from where the command is run, that is */urs/local/spark*.