Name: Hasib Ul Alam
Id: 392449

There are multiple script based on different task.

Task Number	Script Name	output
1.	extract.py	Create CSV files with desired output "data/ACM_1995_2004.csv" and "data/DBLP_1995_2004.csv"
2.	erWithBlocking.py	Create CSV: “data/Matched Entities.csv”
2.	erBruteForce.py	Create CSV : 'data/Matched Entities with Brute Force.csv'
2.	calculatingQualityMeasures.py	Print quality measures based on erWithBlocking and erBruteForce.
2	erClustering.py	Prints ID’s of matched entities in list of sets
3.	spark.py	data/Matched Entities with Spark.csv

To run each file:

```
python filename.py
```

Note: each script is depended on previous script’s output. So each script need to be run chronologically.

Task 1

Code Description:

The provided code is designed to download ACM and DBLP zip files from predefined URLs, extract the data, and organize it into a 'data' folder. Once the text files are extracted, they are cleaned and transformed based on specified constraints, and then loaded into CSV files.

Description of the Text File:

Each publication's information is stored in multiple lines within the text file, with each line starting with a distinct key indicating the type of information it contains. The relevant keys for the task are:

- `#*` --- paperTitle
- `#@` --- Authors
- `#t` ---- Year
- `#c` --- Publication Venue
- `#index` ---- Index ID of the paper
- `#%` ---- ID of references of the paper (multiple lines)

Cleaning/Transforming Process:

The text file is read line by line, and each publication's information is stored in a dictionary called `record`, which is then added to a list called `records`. This list is transformed into a Pandas DataFrame and saved as a CSV file.

For each publication, the code checks the starting string of each line and adds the corresponding information to the `record` dictionary. Additional checks are performed for `publicationVenue` and `yearOfPublication` to ensure they fall within specified constraints (years 1995 to 2004 and venues containing "sigmod" or "vldb"). Publications that do not meet these constraints are not added to the `records` list.

Task 2

Task 2 involves multiple Python files:
- `erWithBlocking.py`: Entity resolution with blocking. It contains the blocking and matching part of ER
- ***`erBruteForce.py`:*** Entity resolution without blocking. It contains matching without blocking part.
- calculatingQualityMeasures.py: contains the quality measures based on previous 2 script
-erClustering.py:. clustering the records founded from ‘erWithBlocking.py` file. Contains the 3rd part(clustering) of ER

Blocking Stage:

For blocking, a dictionary structure was employed. The keys were generated by concatenating the publication year and venue, creating unique identifiers like "SIGMOD2004" or "SIGMOD2001". Each of these keys was associated with a list containing all publications matching that key. For instance:

SIGMOD2004: [publicationRecord_1, publicationRecord_3, ..., publicationRecord_n]

This process was applied to both the ACM and DBLP datasets, resulting in two separate dictionaries. This setup facilitates subsequent matching based on the identical keys extracted from these dictionaries.

Matching Stage With Blocking:

In the matching stage, two dictionaries created in the previous stage were utilized. Using the keys from these dictionaries, a similarity function was employed to compare publications, focusing specifically on their titles. The Jaccard similarity metric was chosen to measure similarity because the goal was to determine whether the words in the titles of two publications were identical or not.

Focusing solely on publication titles for matching, it was concluded that Jaccard similarity sufficed for this purpose. Unlike authors' names, which typically consist of one or two words, publication titles tend to comprise multiple words. Therefore, it was determined that Jaccard similarity was appropriate for comparing titles, without the need for additional metrics like Levenshtein distance or n-grams. These metrics would have been more relevant if authors' names had been included in the comparison process.

The execution time has also been calculated as it was used to see the difference in values. And different threshold values are used to see the execution time and no of entities. It creates a CSV file which in each line contains a recordID(acm publication id + dblp publication id), acmpublication information, dblp publication and the similarity scores between them.

Simirarity Method	Thre shold	Blocking or Brute Force	Execution Time (sec)	No of matched Entities	accuracy	pricision	recall	fScore
Jaccard Similarity	0.70	blocking	2.26	1264	0.9814	1.0	0.9814	0.9906
		No blocking	660	1288				
	0.80	blocking	2.92	585	0.9865	1.0	0.9865	.9932
		No blocking	584.33	593				
	0.90	blocking	1.99	61	0.9839	1.0	0.9839	0.9918
		No blocking	546.36	62				

Table: Quality measures for ER with and without blocking

Matching Stage Without Blocking:
A comparison was done to assess the effectiveness of blocking in entity matching. A script named erBruteForce.py was developed as an alternative to blocking techniques. This script employs a brute force method by iterating through all pairs of records from two files without utilizing blocking. Consequently, it compares each record from one file against every record from another file, resulting in significantly slower performance compared to methods employing blocking.

The *erBruteForce.py* script was executed multiple times with varying Jaccard similarity thresholds to evaluate its performance comprehensively. The output generated from these executions was saved in the file named "data/Matched Entities with Brute Force.csv".

Calculating Quality Measures:
Quality measures has been calculated based on blocking and no blocking. IT has been implemented only for different threshold. From the table above it can be observed high accuracy for all 3 threshold. Although when the threshold is less the algorithm can find more duplicates. Precision is always 1 as the base values are received by bruteforce algorithm. As matching technique(jaccard) is same so with blocking it can only miss values but can not get values which can not be taken by brute force.
The calculation can be seen using the script '*calculatingQualityMeasures.py*'.

Clustering:

The clustering stage has been implemented in the erClustering.py file, focusing on values obtained from Entity Resolution (ER) with blocking. Initially, it reads the file "data/Matched Entities.csv" created by the erWithBlocking.py script. This file includes a recordId column, which represents a combined ID derived from indices of both ACM and DBLP datasets.

During the clustering process, the recordId is split to generate ACM and DBLP publication IDs, forming a tuple for each matched entity. Subsequently, a list of tuples is created for each matched entity. This list serves as input for a connected component algorithm, resulting in a list of sets containing all matched entities. Upon execution, the erClustering.py file displays all publications along with their corresponding matched publications grouped into different clusters.

Task 3

Task 3 involves :
  spark.py file

At first, 2 spark dataframe(dfACM and dfDBLP) has been created from "data/ACM_1995_2004.csv" and "data/DBLP_1995_2004.csv" file.

For blocking stage, a new column named ‘key’ has been added in both dataframe. This “key” column has been created by combining publicationVenue and yearofPublication column from dfACM and dfDBLP dataframe. Afterthat dfACM and dfDBLP is joined into a new dataframe name dfCombined. The join is beased on the ‘key’ column which has been created. This results in a new combined DataFrame (dfCombine) containing potentially matching records

In the matching stage, the Jaccard similarity score is calculated between publication titles from ACM and DBLP datasets using the previously defined function named ‘calculateSimilarity’. calculateSimilarity funtion takes two publication titles as input and returns their Jaccard similarity score. The Jaccard similarity is calculated as the size of the intersection of two sets divided by the size of the union of the sets. This function is applied using a User Defined Function (UDF) in PySpark. A filter on the combined DataFrame based on the threshold similarity score, gives only pairs of records with similarity scores above the given threshold.

Finally, the matched entities are saved to a CSV file named "Matched Entities with Spark.csv" in the "data" directory.