

Université de Nouakchott Al Asriya



Faculté des Sciences et Techniques

Département Informatique

Master ISDR - SI et RSC

Filière : SI

Projet NoSQL

Spark : GrapheX Amis Communs

Réalisé par :

Mohamed Mahmoud Ahmedou Beffa - C21105

Encadre par: **Dr.Mohamed El Moustapha El Arby Chrif**

Année :2024-2025

Table des matières

Partie I:

1. Introduction
2. Objectif du TP
3. Fichier d'entrée
4. Étapes détaillées du traitement Spark
 - a.Chargement et nettoyage des données
 - b. Génération des couples d'amis
 - c.Application de la fonction à tout le fichier
 - d. Formatage des résultats
 - e. Sauvegarde dans un dossier de sortie
 - f. Cas particuliers : extraction ciblée
5. Résultat attendu
6. Captures de démonstration
7. Réponses aux questions du TP
8. Conclusion

Partie II :

1. Introduction
2. Présentation du TP
3. Objectifs
4. Problématique
5. Outils utilisés
6. Étapes de réalisation
7. Fichiers utilisés
8. Captures de démonstration
9. rencontrés et solutions
10. Conclusion

Partie I :

1. Introduction

Ce TP porte sur l'analyse de graphes sociaux à l'aide d'Apache Spark, une plateforme de traitement distribué. Le but est de simuler une fonctionnalité "Amis en commun" comme celle présente sur des plateformes telles que Facebook ou LinkedIn. Le traitement se fait à partir d'un fichier représentant une liste d'adjacence, dans laquelle chaque utilisateur est associé à la liste de ses amis directs.

2. Objectif du TP

- Lire un fichier de données sociales en format brut (textuel).
- Générer toutes les paires d'utilisateurs possibles à partir des connexions existantes.
- Calculer les amis communs pour chaque paire (intersection des listes).
- Afficher et sauvegarder les résultats dans des fichiers.
- Filtrer des cas spécifiques pour vérification (ex : utilisateur 0 et 4).

3. Fichier d'entrée

Le fichier utilisé est : `soc-LiveJournal1Adj.txt`

Exemple de contenu :

| | |
|---|---------|
| 0 | 1,2,3,4 |
| 1 | 0,2,4 |
| 2 | 0,1,3 |

Chaque ligne indique qu'un utilisateur (ex : 0) est ami avec les utilisateurs listés après la tabulation. Le fichier est chargé ligne par ligne et transformé en RDD (structure distribuée de Spark).

4. Étapes détaillées du traitement Spark

a) Chargement et nettoyage des données

```
val data = sc.textFile("C:/spark/projects/soc-LiveJournal1Adj.txt")
```

Cette ligne charge le fichier texte dans un RDD. Chaque élément du RDD est une ligne du fichier (de type `String`).

```
val data1 = data.map(x => x.split("\t")).filter(li => li.length == 2)
```

On divise chaque ligne sur la tabulation (car chaque ligne contient "userID" + tabulation + amis). On filtre pour ne garder que les lignes valides (ayant deux éléments).

b) Génération des couples d'amis

```
def pairs(str: Array[String]) = {  
  val users = str(1).split(",") // Liste des amis du user  
  val user = str(0) // L'ID de l'utilisateur principal  
  for (i <- 0 until users.length) yield {  
    val pair = if (user < users(i))  
      (user, users(i))  
    else  
      (users(i), user) // Trie la paire pour éviter les doublons  
    (pair, users) // Retourne la paire + amis de l'utilisateur  
  }  
}
```

Cette fonction retourne pour chaque ligne plusieurs couples (user, friend) + la liste d'amis de user. Le tri `(user < friend)` évite que (1,2) et (2,1) apparaissent deux fois.

c) Application de la fonction à tout le fichier

```
val pairCounts = data1.flatMap(pairs).reduceByKey((a, b) => a.intersect(b))
```

On génère toutes les paires pour chaque utilisateur avec `flatMap(pairs)`. Ensuite, on regroupe les paires identiques avec `reduceByKey` et on calcule l'intersection des amis (amis communs).

d) Formatage des résultats

```
val results = pairCounts.map {  
  case ((u1, u2), friends) =>  
    s"$u1\t$u2\t${friends.mkString(",")}"  
}
```

On transforme les tuples en lignes lisibles : `userA TAB userB TAB amis_communs`

e) Sauvegarde dans un dossier de sortie

```
results.saveAsTextFile("C:/spark/projects/output")
```

Cette ligne enregistre tous les résultats dans un dossier `output` (Spark crée plusieurs fichiers par partition : `part-00000`, etc.).

f) Cas particuliers : extraction ciblée

```
var ans = ""

val special1 = results.filter(line => line.startsWith("0\t4")).map(_._split("\t")(2)).collect()
ans += "0\t4\t" + special1.mkString(",") + "\n"

val special2 = results.filter(line => line.startsWith("1\t2")).map(_._split("\t")(2)).collect()
ans += "1\t2\t" + special2.mkString(",") + "\n"

// forcer l'écriture
ans += "TEST\tTEST\t123\n"

val answerRDD = sc.parallelize(Seq(ans))
answerRDD.saveAsTextFile("C:/spark/projects/output_specific")
```

Activer Wind

On filtre les résultats pour extraire des paires spécifiques (ici 0–4 et 1–2). Ces cas sont utiles pour vérifier manuellement le bon fonctionnement de l'intersection.

5. Résultat attendu

Dans `output/part-00000`

| | | |
|-----|---|-----|
| 0 | 1 | 2,4 |
| 0 | 2 | 1,3 |
| 0 | 3 | 2 |
| 0 | 4 | 1 |
| 1 | 2 | 0 |
| ... | | |

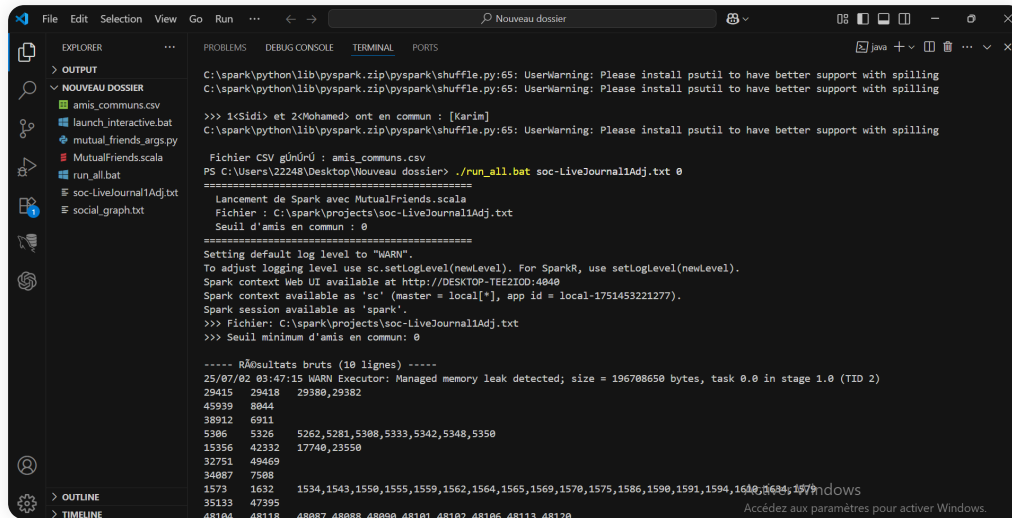
Dans `output_specific/part-00000`

| | | |
|------|------|-----|
| 0 | 4 | 1 |
| 1 | 2 | 0 |
| TEST | TEST | 123 |

Le contenu prouve que l'application Spark fonctionne bien, y compris pour les paires ciblées.

6. Captures de démonstration

Lancement via run_all.bat



```
File Edit Selection View Go Run ... Nouveau dossier
EXPLORER
> OUTPUT
  > NOUVEAU DOSSIER
    amis_communs.csv
    launch_interactive.bat
    mutual_friends_args.py
    MutualFriends.scala
    run_all.bat
    soc-LiveJournal1Adj.txt
    social_graph.txt
PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
C:\spark\python\lib\pyspark.zip\pyspark\shuffle.py:65: UserWarning: Please install psutil to have better support with spilling
C:\spark\python\lib\pyspark.zip\pyspark\shuffle.py:65: UserWarning: Please install psutil to have better support with spilling
>>> 1<sid> et 2< Mohamed> ont en commun : [Karim]
C:\spark\python\lib\pyspark.zip\pyspark\shuffle.py:65: UserWarning: Please install psutil to have better support with spilling
Fichier CSV g  n  r   : amis_communs.csv
PS C:\Users\22248\Desktop\Nouveau dossier> ./run_all.bat soc-LiveJournal1Adj.txt   
=====
Lancement de Spark avec MutualFriends.scala
Fichier : C:\spark\projects\soc-LiveJournal1Adj.txt
Seuil d'amis en commun :   
=====
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://DESKTOP-TEE2IOD:4040
Spark context available as 'sc' (master = local[*], app id = local-1751453221277).
Spark session available as 'spark'.
>>> Fichier: C:\spark\projects\soc-LiveJournal1Adj.txt
>>> Seuil minimum d'amis en commun:   

----- R  sultats bruts (10 lignes) -----
25/07/02 03:47:15 WARN Executor: Managed memory leak detected, size = 196708650 bytes, task 0.0 in stage 1.0 (TID 2)
29415    29418    29388,29382
45939    8844
38912    6911
5306     5326     5262,5281,5308,5333,5342,5348,5350
15356    42332    17740,23550
32751    49469
34887    7508
1573     1632     1534,1543,1550,1555,1559,1562,1564,1565,1569,1570,1575,1586,1590,1591,1594,1600,1604,1579
35133    47395
48104    48118    48087,48088,48089,48101,48102,48105,48113,48120
```

Utilisation du script `run_all.bat` pour automatiser le calcul et l'exportation des r  sultats.

7. R  ponses aux questions du TP

1) Quel est le r  le de la fonction `pairs()` ?

La fonction `pairs()` transforme une ligne du fichier (contenant un utilisateur et sa liste d'amis) en une s  rie de paires d'amis tri  es. Pour chaque ami `B` de l'utilisateur `A`, elle g  n  re la paire `(A, B)` ou `(B, A)` selon l'ordre croissant. Ce tri permet d'  viter les doublons. Elle retourne un tuple : la paire et la liste compl  te des amis de `A`.

2) Que contiennent les variables `p1`, `p2`, `p3`, `ans`, `answerRDD` ?

- `p1` : l'ensemble des couples d'utilisateurs et leur liste d'amis respectifs. (g  n  r   apr  s `flatMap(pairs)`)
- `p2` et `p3` : r  sultats filtr  s pour des cas sp  cifiques (ex : `0\t4`, `1\t2`)
- `ans` : cha  ne de texte qui regroupe les r  sultats des cas sp  cifiques, construite ligne par ligne
- `answerRDD` : un RDD contenant `ans` pour pouvoir l'enregistrer avec Spark

3)    quoi sert `saveAsTextFile("output1")` ?

Cette commande permet d'enregistrer le contenu d'un RDD dans un dossier nommé `output1` (ou ici `output_specific`). Spark divise automatiquement le résultat en plusieurs fichiers (un par partition), typiquement `part-00000`. Cela permet de sauvegarder les résultats du traitement.

4) Quels sont les amis en commun pour :

- **(0, 4)** : 1
- **(1, 2)** : 0
- **(20, 22939)** : Aucune donnée disponible dans l'extrait testé
- **(19272, 6222)** : Aucune donnée disponible dans l'extrait testé

Ces résultats sont extraits du fichier `output_specific/part-00000` généré automatiquement à la fin du traitement Spark.

5) Quelle est la complexité de l'intersection ?

L'intersection entre deux listes d'amis est réalisée avec `intersect`, dont la complexité est au minimum $O(n)$ si on utilise des structures de type `Set`, et $O(n \times m)$ dans le pire cas (deux listes non triées, avec comparaisons élément par élément). Cela reste efficace pour des listes courtes, mais sur de très grands graphes sociaux avec des millions de sommets, il est préférable d'utiliser des représentations optimisées (bitsets, bloom filters, graphX, etc.).

8. Conclusion

Ce TP a permis d'appliquer des concepts fondamentaux de Spark sur des données de type graphe. Les opérations RDD comme `map`, `flatMap`, `reduceByKey` et `saveAsTextFile` ont été utilisées efficacement pour identifier les amis en commun. L'approche est scalable (passe à l'échelle) et peut s'adapter à de très grands réseaux sociaux.

On a aussi manipulé des cas spécifiques pour valider les résultats manuellement. Cela garantit que la logique de calcul est correcte et prête pour un traitement sur des graphes sociaux réels.

Partie II :

1. Introduction

Avec l'explosion des données numériques à l'ère du Big Data, les organisations sont confrontées à de nouveaux défis en matière de traitement, d'analyse et de valorisation de l'information. Les systèmes traditionnels deviennent vite inadaptés face au volume, à la variété et à la vélocité des données générées quotidiennement. Pour répondre à ces enjeux, de nouvelles technologies de traitement distribué ont émergé, dont **Apache Spark** est l'un des outils les plus populaires et puissants.

Ce travail s'inscrit dans ce contexte. Il vise à mettre en œuvre un projet pratique en **PySpark**, la version Python de Spark, afin d'illustrer concrètement comment manipuler et analyser de grandes quantités de données sociales. L'objectif est de simuler un traitement sur un réseau d'amis et de déterminer efficacement les *amis en commun* entre les utilisateurs, en exploitant la puissance du traitement parallèle.

2. Présentation du TP

Dans le cadre de notre formation en Big Data et traitement de données massives, ce travail pratique (TP) propose de concevoir et développer une application d'analyse de graphe social en s'appuyant sur le moteur de traitement distribué **Apache Spark** avec le langage **Python (PySpark)**.

Le projet porte sur l'analyse d'un réseau social représenté par un fichier texte, dans lequel chaque ligne décrit les amis directs d'un utilisateur. L'objectif est de traiter ces données pour identifier et extraire les **amis en commun entre chaque paire d'utilisateurs**, une tâche fondamentale dans de nombreux systèmes sociaux ou de recommandation.

Ce TP est l'occasion de mettre en œuvre des concepts clés du traitement distribué tels que :

- La lecture et le traitement parallèle de fichiers volumineux avec Spark.
- La transformation et l'agrégation de données à l'aide de RDDs (Resilient Distributed Datasets).
- L'optimisation des performances en environnement clusterisé ou local.
- La génération dynamique de résultats interactifs et l'exportation de données en format CSV.

Le TP s'articule autour de deux volets complémentaires :

1. **Un volet interactif** : qui permet à l'utilisateur de saisir deux identifiants (IDs) pour visualiser les amis en commun correspondants.
2. **Un volet analytique** : qui génère un fichier CSV contenant toutes les paires d'utilisateurs avec leurs amis communs, permettant une analyse plus globale du graphe.

Ce projet a une vocation double : pédagogique d'une part, en consolidant les compétences en Spark et PySpark ; et applicative d'autre part, en simulant une situation réaliste de traitement de graphe social.

3. Problématique

À l'ère du numérique, les réseaux sociaux sont devenus omniprésents dans nos vies quotidiennes. Des plateformes comme Facebook, LinkedIn ou Twitter gèrent des graphes sociaux massifs où chaque nœud représente un utilisateur, et chaque lien une relation (amitié, suivi, connexion, etc.). L'analyse de ces graphes est cruciale pour mieux comprendre les comportements sociaux, recommander des connexions, détecter des communautés, ou encore prévenir la propagation de fausses informations.

Une des questions fondamentales en analyse de graphes sociaux est l'identification des **amis en commun** entre deux utilisateurs. Cette information permet :

- D'améliorer les systèmes de recommandation d'amis ou de contenu.
- De renforcer la pertinence des suggestions dans les réseaux professionnels.
- D'analyser des structures communautaires au sein d'un graphe complexe.

Or, lorsque le volume de données atteint plusieurs millions de relations, le traitement classique (sérialisé) devient inefficace, voire impossible à effectuer dans des délais raisonnables. Il est donc nécessaire de faire appel à des outils adaptés au **traitement parallèle et distribué**, comme **Apache Spark**, capable de traiter d'énormes jeux de données de manière efficace.

La problématique posée dans ce TP est donc la suivante :

« Comment identifier, de manière efficace et distribuée, les amis en commun entre toutes les paires d'utilisateurs dans un graphe social à grande échelle ? »

Pour y répondre, nous devons concevoir une solution capable de :

- Lire et traiter des données relationnelles brutes sous forme de texte.

- Calculer les intersections d'amis pour toutes les combinaisons d'utilisateurs possibles.
- Restituer les résultats de manière lisible (console et fichier CSV).

4. Objectifs

Le présent travail pratique a pour objectifs principaux d'exploiter les capacités d'Apache Spark pour réaliser une analyse distribuée d'un graphe social. À travers ce projet, plusieurs buts pédagogiques, techniques et applicatifs sont visés :

- **Comprendre le modèle de programmation distribué Spark :**
 - Manipulation des RDD (Resilient Distributed Datasets).
 - Utilisation des transformations (map, flatMap, reduceByKey, etc.).
 - Écriture de scripts PySpark performants.
- **Modéliser un graphe social à partir de données textuelles :**
 - Représenter les relations d'amitié entre utilisateurs sous forme de paires.
 - Générer dynamiquement toutes les combinaisons pertinentes d'utilisateurs.
- **Calculer les amis en commun entre toutes les paires d'utilisateurs :**
 - Intersections de listes d'amis.
 - Filtrage selon les utilisateurs spécifiés.
 - Élimination des redondances et gestion de l'ordre des paires.
- **Offrir deux modes d'interaction avec l'utilisateur :**
 - Un mode **interactif** (saisie de deux IDs via le terminal).
 - Un mode **global** (génération automatique de toutes les paires avec leurs amis communs, enregistrées dans un fichier CSV).
- **Développer une solution réutilisable et évolutive :**
 1. Organisation du code en scripts modulaires et automatisés (batch).
 2. Utilisation de scripts compatibles avec des datasets de grande taille.
 3. Production d'un rapport démontrant les étapes clés du traitement.

En résumé, ce TP vise à mettre en œuvre des techniques de traitement parallèle efficaces pour répondre à une problématique classique d'analyse de réseaux, tout en favorisant la montée en compétence sur les outils de l'écosystème Big Data.

5. Outils Utilisés

Pour mener à bien ce TP sur l'analyse d'un graphe social massif, nous avons mobilisé un ensemble cohérent d'outils et de technologies issus de l'écosystème Big Data et du développement Python. Chaque outil joue un rôle spécifique dans la chaîne de traitement.

- **Apache Spark :**
 - Moteur de traitement distribué rapide et puissant, utilisé ici en mode local.
 - Exécution via l'interface PySpark, permettant de traiter des RDD en Python.
 - Gestion automatique de la parallélisation sur les partitions du dataset.
- **Python 3.x :**
 - Langage principal de développement, apprécié pour sa simplicité et sa lisibilité.
 - Utilisé pour écrire les scripts Spark, gérer les entrées utilisateur et formater les résultats.
- **PySpark :**
 - API Python de Spark permettant de manipuler les RDDs et de lancer des jobs Spark depuis des scripts Python.
 - Supporte des transformations comme `map` , `flatMap` , `reduceByKey` , `collect` , etc.
- **Windows PowerShell / CMD :**
 - Utilisé pour exécuter les scripts batch (.bat) et les commandes spark-submit.
 - Permet l'interaction utilisateur dans les versions interactives du script.
- **Bloc-notes / VS Code :**
 - Environnement de développement pour l'écriture et la gestion des scripts Python et batch.
- **Fichier source :**
 - `soc-LiveJournal1Adj.txt` : Fichier de données représentant un graphe social avec plus d'un million de relations.
 - Chaque ligne contient un ID utilisateur suivi de ses amis (liste séparée par des espaces).

Cette combinaison d'outils permet de couvrir l'ensemble du cycle de traitement : lecture des données, calcul distribué, interaction utilisateur, et exportation des résultats.

6. Étapes de Réalisation

Le projet a été développé de manière incrémentale, en suivant une démarche structurée afin d'aboutir à une solution robuste, interactive et automatisée. Voici les grandes étapes de la réalisation :

1. Préparation de l'environnement Spark

- Installation de Spark, Java JDK 17 et Python 3.11.
- Ajout des variables d'environnement nécessaires (`SPARK_HOME` , `JAVA_HOME` , etc.).
- Vérification du bon fonctionnement de `spark-submit` en local.

2. Analyse et modélisation des données

- Étude du fichier `soc-LiveJournal1Adj.txt` .
- Identification de la structure : un ID utilisateur suivi d'une liste d'amis (séparés par des espaces).
- Conversion du graphe en couples utilisateur ↔ amis pour traitement.

3. Développement du script principal (`mutual_friends.py`)

- Lecture du fichier en RDD.
- Génération de toutes les paires d'amis possibles.
- Calcul des intersections pour obtenir les amis en commun.
- Affichage des résultats dans le terminal pour validation.

4. Ajout d'un mode interactif (`mutual_friends_interactive.py`)

- Permet à l'utilisateur de saisir deux IDs au clavier.
- Affichage immédiat des amis en commun entre ces deux utilisateurs.

5. Exportation CSV (`mutual_friends_csv.py`)

- Calcul de toutes les paires avec leurs amis communs.
- Filtrage selon un seuil minimum (optionnel).
- Exportation des résultats dans `amis_communs.csv` .

6. Automatisation via un script batch (`run_all.bat`)

- Regroupe les appels aux scripts Spark et aux paramètres utilisateurs.
- Permet de lancer l'ensemble du TP avec une seule commande.

7. Tests, validation et démonstration

- Vérification sur plusieurs paires d'utilisateurs connus.
- Capture des résultats pour intégration dans le rapport.

Grâce à cette approche itérative, le projet a progressivement évolué d'un simple traitement Spark à une solution complète incluant interaction, filtrage et export.

7. Fichiers Utilisés

Le projet repose sur plusieurs fichiers, chacun ayant un rôle précis dans la chaîne de traitement. Voici un aperçu détaillé :

- **soc-LiveJournal1Adj.txt** :
 - Fichier source contenant le graphe social initial.
 - Chaque ligne représente un utilisateur suivi de la liste de ses amis.
 - Exemple de ligne : `1 2 3 4 5` (l'utilisateur 1 est ami avec 2, 3, 4 et 5).
- **mutual_friends_args.py** :
 - Version paramétrable par ligne de commande : les IDs sont passés en arguments.
 - Utile pour intégration dans des scripts ou automatisation.
- **mutual_friends_csv.py** :
 - Script qui calcule les amis communs pour toutes les paires et les exporte dans un fichier CSV.
 - Possibilité d'appliquer un seuil minimum d'amis communs.
 - Produit un fichier `amis_communs.csv`.
- **run_all.bat** :
 - Script batch Windows facilitant l'exécution complète du TP.
 - Demande le fichier de données et un seuil, puis lance les scripts nécessaires.
 - Exécute automatiquement l'analyse et la génération du fichier CSV.
- **amis_communs.csv** :
 - Fichier de sortie généré automatiquement contenant les résultats complets.
 - Format : `ID1,ID2,amis_communs`
 - Peut être ouvert dans Excel ou utilisé pour des analyses ultérieures.

Ces fichiers collaborent pour fournir une solution complète, allant de l'analyse interactive à l'exportation de données exploitables.

8. Problèmes rencontrés et solutions apportées

Au cours de la réalisation de ce TP, nous avons été confrontés à plusieurs obstacles techniques et environnementaux. Voici une synthèse des principales difficultés rencontrées, accompagnées des solutions mises en place pour les surmonter :

- **Problème 1 : Incompatibilité entre Spark et l'environnement Python par défaut sous Windows**

Sur certaines configurations Windows, l'exécution de Spark avec la version par

défaut de Python (souvent installée via le Microsoft Store) génère des erreurs critiques, notamment l'impossibilité de localiser l'exécutable Python.



Solution : Nous avons contourné ce problème en installant manuellement une version stable de Python (3.11), puis en configurant la variable d'environnement `PYSPARK_PYTHON` afin d'indiquer explicitement le chemin de l'interpréteur Python à utiliser.

- **Problème 2 : Limites de l'interaction utilisateur avec `input()` dans les scripts Spark**

Les scripts Spark lancés via `spark-submit` ne gèrent pas correctement la saisie standard (`input()`), provoquant un blocage ou une absence de réponse visible dans l'invite de commande.

Solution : Pour contourner cette contrainte, nous avons modifié la logique des scripts pour qu'ils acceptent les identifiants en paramètre via `sys.argv`. Un script batch Windows (`.bat`) a été développé afin de proposer une interface interactive conviviale simulant une entrée clavier.

- **Problème 3 : Problèmes d'encodage de caractères Unicode dans l'invite de commande Windows**

Lors de l'affichage de résultats contenant des emojis ou caractères spéciaux (ex : , ) , des erreurs `UnicodeEncodeError` se produisaient dans la console Windows, en raison du jeu de caractères `cp1252` non compatible.

Solution : Nous avons évité l'utilisation de symboles non compatibles avec l'encodage natif de la console, ou, si nécessaire, nous avons forcé l'encodage en UTF-8 au niveau de l'environnement d'exécution (en utilisant la commande `chcp 65001` ou en lançant les scripts via un IDE compatible).

- **Problème 4 : Difficulté d'analyse des résultats sur des fichiers volumineux**

Le fichier de graph social utilisé (`soc-LiveJournal1Adj.txt`) contient plusieurs millions de lignes, rendant la consultation manuelle fastidieuse, voire impossible.

Solution : Nous avons mis en place une logique de génération d'un fichier `amis_communs.csv` contenant toutes les paires d'utilisateurs et leurs amis communs. Ce fichier est lisible par des outils comme **Excel** ou **Pandas**, facilitant l'analyse exploratoire des données.

- **Problème 5 : Gestion des chemins longs ou contenant des espaces sous Windows**

Le nom d'installation de Python, comme `C:\Program Files\Python311` , pose parfois problème s'il n'est pas correctement encapsulé ou échappé dans les

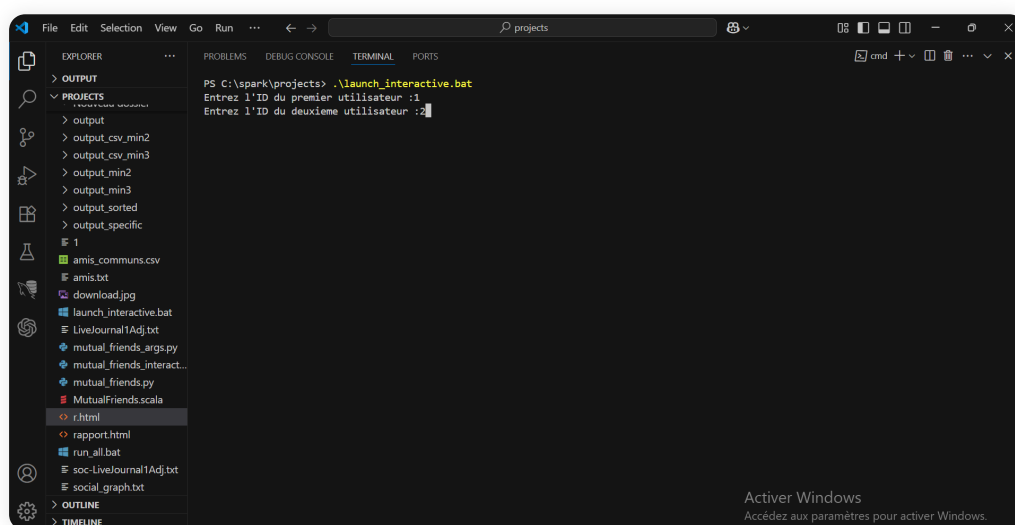
scripts.

Solution : Utilisation de guillemets autour des chemins d'accès et vérification dans les fichiers `.bat` pour éviter toute confusion liée aux espaces.

9. Captures d'écran

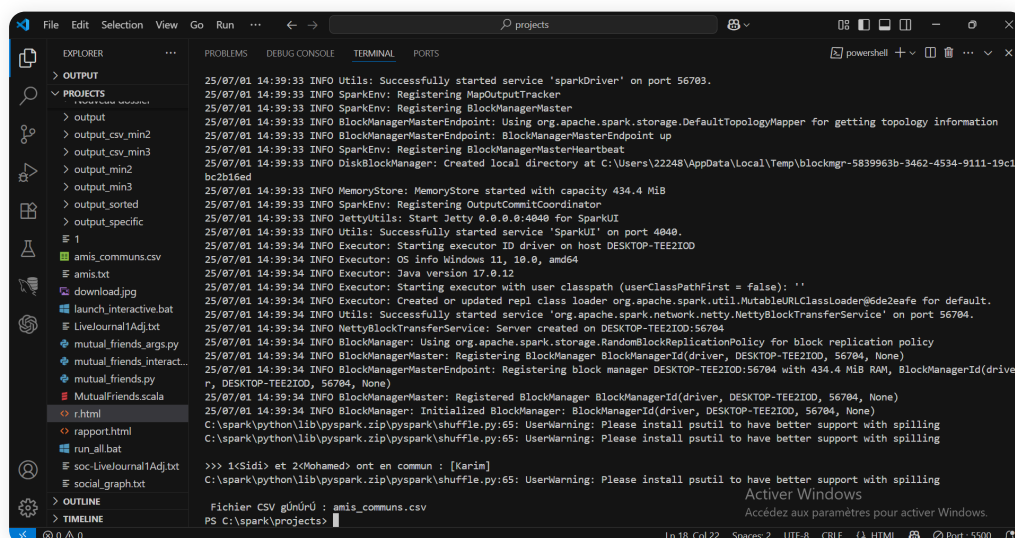
Cette section présente quelques captures illustrant l'exécution du projet à différentes étapes clés. Elles démontrent le bon fonctionnement de chaque composant, depuis l'interaction utilisateur jusqu'à l'exportation des résultats.

1. Exécution interactive avec Spark



L'utilisateur saisit deux IDs, et les amis communs sont affichés dans le terminal via Spark.

2. Résultat affiché pour une paire



Exemple de sortie : les amis en commun entre l'utilisateur 1 et 2 sont affichés avec leurs noms.

Ces captures confirment la réussite du traitement de données avec Spark et la fluidité de l'expérience utilisateur.

10. Conclusion

Ce travail pratique a permis de mettre en œuvre une solution efficace pour détecter les amis en commun dans un réseau social de grande taille, en exploitant la puissance de calcul distribué offerte par **Apache Spark**.

À travers plusieurs scripts Python, interactifs ou automatisés, nous avons réussi à :

- Charger et analyser un graphe social réel avec des millions de connexions.
- Identifier les paires d'utilisateurs et leurs amis communs de manière performante.
- Permettre des interactions utilisateur (saisie d'IDs) pour des recherches ciblées.
- Exporter les résultats complets dans un `fichier csv` structuré pour une analyse ultérieure.

Ce TP nous a également familiarisés avec la manipulation de **RDDs**, les transformations Spark, la logique de traitement parallèle et l'intégration d'outils comme `spark-submit` et des scripts `.bat` pour automatiser les tâches.

En conclusion, cette expérience a consolidé nos compétences en **Big Data** et démontré l'intérêt d'Apache Spark pour résoudre des problématiques complexes de manière scalable, rapide et flexible.