Rapport

TP1 Big Data Analytics

auteurs:
Ahmad HARKOUS
Thomas FERMELI FURIC

mars, 2024

Table des matières

1 Exercice 1

1.1 Question 4

Le code pour cette question se trouve au chemin suivant :

ex1/alph_occ/q4.py

Dans le **mapper**, pour chaque mot d'une ligne et si ce mot n'est constitué que de lettres, nous renvoyons un tuple avec comme clé la première lettre du mot en majuscule et comme valeur 1. Le choix de la casse pour cette première lettre est arbitraire, le but étant d'homogénéiser les clés afin qu'elles se réunissent. Dans le **reducer**, nous allons donc sommer toutes les valeurs reçues pour chacune des clés, ce qui nous donnera le total des fois où chaque lettre débutait un mot.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex1/alph_occ/q4_output.txt

1.2 Question 5

Le code pour cette question se trouve au chemin suivant :

ex1/anagramme/anagramme.py

Dans le **mapper**, nous commençons par retirer la ponctuation de chaque ligne, séparer les mots, puis pour chaque mot de longueur supérieure à un et ne contenant que des lettres, nous allons réordonner ses lettres dans l'odre alphabétique et en minuscule, de telle sorte que les anagrammes aient la même clé et soient regroupés. Dans le **reducer**, pour chacune des clés, nous rangeons ses valeurs dans un set afin de supprimer les doublons, puis nous renvoyons le set si ça longeur est supérieure à un, ce qui signifie que plusieurs mots comportant exactement les mêmes lettres ont été trouvés.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex1/anagramme/output_anagramme.txt

1.3 Question 6

Le code pour cette question se trouve au chemin suivant :

ex1/palindrome/palindrome.py

Dans le **mapper**, nous retirons la ponctuation de chaque ligne, séparons les mots et pour chaque mot de longeur supérieure à un et ne contenant que des lettres, nous vérifions s'il est un palindrome à l'aide d'une fonction annexe. Si c'est le cas, le mapper renvoie un tuple constitué du palindrome en clé et d'une valeur quelconque. Le **reducer** renvoie à son tour le tuple tel qu'il la reçu car aucune opération n'est nécessaire après le travail du mapper, et de potentiels doublons ont déjà été supprimés lorsque les tuples sont regroupés par clés.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex1/palindrome/output_palindrome.txt

2 Exercice 2

Le code pour cet exercice se trouve au chemin suivant :

ex2/baseball.py

Dans le **mapper**, le programme effectue les opérations suivantes pour chaque ligne d'entrée du fichier de données :

- **Extraction des informations** : La ligne est divisée en trois parties (nom, équipe, amis) à l'aide de la méthode split. La liste des amis est ensuite créée en séparant les noms avec des virgules, et chaque nom est nettoyé de tout espace en utilisant la méthode strip.
- **Émission de données intermédiaires** : Les données extraites sont émises sous forme de paires clé-valeur. La clé est l'équipe favorite extraite de la ligne, et la valeur est un tuple contenant le nom de la personne et la liste de ses amis. **(favourite_team, (person_name**

,person_friends))

Le mapper nous permet, en quelque sorte, de récupérer la liste des personnes qui ont la même équipe favorite.
Par exemple après le shuffle and sort on a : **("Cardinals" , [(person1_name , [person1_friends]) , (person2_name , [person2 _friends])])**
Cela équivaut à dire que 'person1' et 'person2' ont comme équipe favorite les Cardinals.

Dans le reducer, on récupère premièrement la liste des noms des personnes qui ont en commun l'équipe favorite X..

Par exemple ("Cardinals" , [(person1_name , [person1_friends]) , (person2_name , [person2_friends])])
→ **Cardinals_fans_names = [person1_name , person2_name]**.

Par la suite on parcours la liste des personne qui ont le meme equipe favorite , dans ce cas : [(person1_name , [person1_friends]) , (person2_name , [person2_friends])] .
On récupère le nom de la personne, puis la liste de ses amis. Ensuite, on parcourt la liste de ces amis et on ne récupère que les noms des amis qui sont présents dans la liste des fans, dans ce cas Cardinals_fans_names.

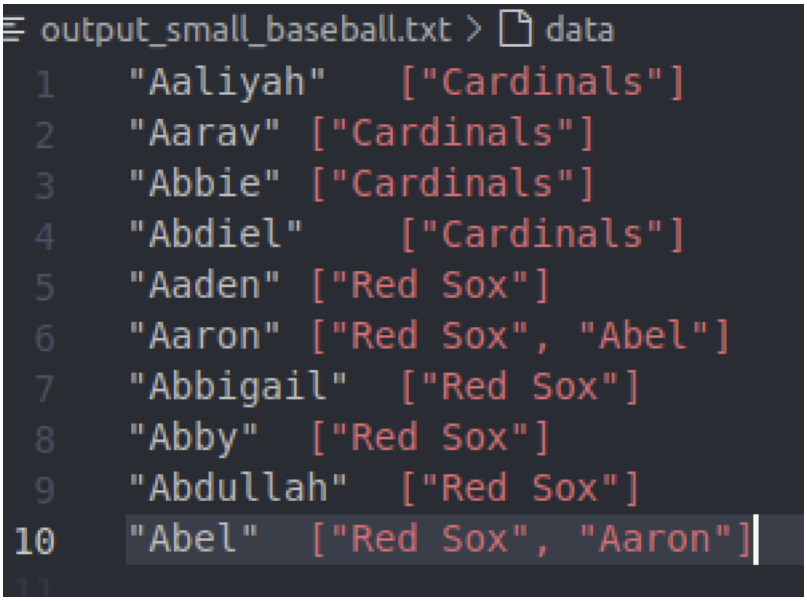
Un exemple du déroulement de l'algorithme :

1. **Input_reducer** = ("$Cardinals$", [($person1\_name$, [$person1\_friends$]) , ($person2\_name$, [$person2\_friends$])])
2. **Équipe favorite** = Cardinals
3. **Cardinals_fans_names** = [$person1\_name, person2\_name$]
4. Pour chaque élément de **Input_reducer**
   Stockage du nom de la personne et de la liste de ses amis
   - **Ex: Itération 1 :** name = person1_name, **friends_list** = person1_friends
   - Pour chaque élément de **friends_list** on récupère uniquement ceux qui sont présents dans la liste **Cardinals_fans_names**.
     * friends_list = [$person2\_name, person3\_name$]
     * person2_name est dans la liste des Cardinals_fans_names et non pas person3_name.

    - Donc, ajout de person2 à la liste des amis de person1 qui ont la même équipe favorite : **same_fav_team** = [*person2_name*]
  - On retourne (person1_name, (Cardinals, same_fav_team))
5. ..... iteration n

On a testé notre algorithme sur les 10 premières lignes des données, pour vérifier sa performance et son bon fonctionnement.
Voici une capture d'écran du résultat.

Abel supporte l'équipe des Red Sox, et de même pour Aron.



Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex2/output_small_baseball.txt
ex2/output_baseball.txt

3 Exercice 3

3.1 Question 1

Le code pour cette question se trouve au chemin suivant :

ex3/total_amount/ex3-q1.py

Dans le **mapper**, nous séparons tous les champs de chaque ligne, puis en ignorant la première ligne qui est la seule à ne pas avoir un identifiant numérique comme premier champ, nous renvoyant des tuples composés de la date année/mois en clé et le montant payé en valeur. Dans le **reducer**, les totaux payés pour chaque année/mois ont été groupés et nous pouvons simplement les sommer. Nous renvoyons alors la date année/mois en clé et le montant total payé durant ce mois en valeur.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex3/total_amount/output_ex3-q1.txt

3.2 Question 2

Le code pour cette question se trouve au chemin suivant :

ex3/mean-price/ex3-q2.py

Dans le **mapper**, pour chaque ligne sauf la première, nous séparons les champs et renvoyons des tuples avec une clé quelconque mais la même pour toutes les lignes car nous souhaitons regrouper tous nos enregistrements, et comme valeur un autre tuple composé du montant payé pour u enregistrement et la durée de stationnement associée. Dans le **reducer**, nous additionnons alors tous les montants payés et toutes les durées de stationnement afin de calculer le prix moyen par heure de stationnement. Nous renvoyons ensuite le résultat obtenons à cette opération.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex3/mean-price/output_ex3-q2.txt

3.3 Question 3

Le code pour cette question se trouve au chemin suivant :

ex3/proportion-PC-CB/ex3-q3.py

Dans le mapper, pour chaque ligne sauf la première, nous récupérons le champ correspondant au moyen de paiement et l'utilisons comme clé du tuple à renvoyer, avec comme valeur 1. Dans un premier reducer, nous sommons simplement les valeurs obtenues pour chaque clé ce qui nous donne le nombre d'enregistrements pour chaque moyen de paiement. Nous renvoyons les résultats obtenus en tant que valeur avec la même clé quelconque car nous souhaitons regrouper les deux sommes dans une même opération reduce. C'est ce que nous faisons dans le second reducer, nous récupérons les valeurs et calculons simplement le ratio de Paris Carte sur les Cartes Bancaires.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex3/proportion-PC-CB/output_ex3-q3.txt

3.4 Question 4

Le code pour cette question se trouve au chemin suivant :

ex3/mean-time/ex3-q4.py

Dans le mapper, pour chaque ligne sauf la première, nous renvoyons un tuple avec comme clé le type d'usager et comme valeur la durée de stationnement. Dans le reducer, nous sommons toutes les durées de stationnement et comptons le nombre d'enregistrements. Grâce à cela, pour chaque type d'usager reçu, nous pouvons calculer puis renvoyer la moyenne des durées de stationnement.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex3/mean-time/output_ex3-q4.txt

3.5 Question 5

Le code pour cette question se trouve au chemin suivant :

ex3/most-lucrative/ex3-q5.py

Dans le mapper, nous regroupons les enregistrements avec comme clé l'identifiant numérique du parcmètre et comme valeurs les différents revenus généré par chaque parcmètre. Dans un premier reducer, nous calculons le total des revenes générés par chaque parcmètre et nous regroupons tous les tuples ensemble avec une clé quelconque. Enfin, dans le dernier reducer, nous déduisons le parcmètre avec le total de revenus le plus élevé et renvoyons son identifiant ainsi que le montant qu'il a généré.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex3/most-lucrative/output_ex3-q5.txt

3.6 Question 6

Le code pour cette question se trouve au chemin suivant :

ex3/10-lowest-highest-mean-price/ex3-q6.py

Dans le mapper, nous regroupons les enregistrements avec comme clé commune l'identifiant du parcmètre et comme valeurs associées le prix du stationnement à l'heure de chaque enregistrement pour chacun des parcmètres. Dans le premier reducer, pour chaque parcmètre, nous calculons le prix moyen par heure de stationnement. Dans le dernier reducer, nous déduisons les dix valeurs les plus basses et les plus hautes dans les moyennes de prix de stationnement et nous les renvoyons.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex3/10-lowest-highest-mean-price/output_ex3-q6.txt

3.7 Question 7

Le code pour cette question se trouve au chemin suivant :

ex3/least-used/ex3-q7.py

Dans le mapper, nous regroupons les durées de stationnement avec chacun des identifiants de parcmètres associés. Dans un premier reducer, pour chaque identifiant de parcmètre, nous sommons les durées de stationnement pour obtenir le temps total de stationnement pour chaque parcmètre Nous regroupons tous les enregistrements grâce à une clé quelconque pour pouvoir opérer sur tous à la fois. Alors, dans un dernier reducer, nous déduisons simplement le parcmètre dont la durée totale de stationnement est la plus faible.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex3/least-used/output_ex3-q7.txt

4 Exercice 4

4.1 Question 1

Le code pour cette question se trouve au chemin suivant :

ex4/q1/q1.py

Dans la fonction **mapper**, chaque ligne du jeu de données est traitée individuellement. On extrait la valeur de latitude de la ligne, en prenant en compte les données manquantes. Si la latitude est manquante, le programme émet un compteur de 1 pour l'hémisphère 'vide'. Sinon, la latitude est convertie en nombre et vérifiée pour déterminer si elle se trouve dans l'hémisphère nord ou sud. Des compteurs de 1 sont émis en conséquence. La fonction **reducer** agrège ensuite les résultats partiels reçus du mappage. Elle additionne simplement les compteurs pour chaque catégorie d'hémisphère ('vide', 'nord' ou 'sud'). **Le résultat final** est émis avec une clé indiquant la catégorie d'hémisphère et la somme des compteurs, **23955 stations sont en hemisphère nord et 4489 en hemisphère sud**

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex4/q1/q1.output.txt

4.2 Question 2

Le code pour cette question se trouve au chemin suivant :

ex4/q2/q2.py

Dans la fonction **mapper**, chaque ligne du jeu de données est analysée individuellement. On extrait d'abord le code du pays à partir de la ligne. Si le code du pays n'est pas vide et correspond à 'FR' (France), on extrait ensuite la latitude, l'altitude et le nom de la station. Les données sont émises sous forme de tuple **('FR', (nom, latitude, altitude))** seulement si au moins l'une des informations de latitude ou d'altitude est présente. La fonction **reducer** reçoit ces tuples pour la France. Elle crée deux listes distinctes : une pour les latitudes valides et une pour les altitudes valides. Ensuite, elle trouve la station avec la plus grande altitude, la station la plus au sud, et la station la plus au nord en utilisant les fonctions **max** et **min** sur les listes appropriées. Les résultats sont émis avec les noms des stations associées.
Le résultat final donne que

- "La station FR qui a la plus grande altitude est: " ["PIC DU MIDI", 2883.0]
- "La station FR qui se trouve le plus au sud est: " ["KERGUELEN IS", -48.667]
- "La station FR qui se trouve le plus au nord est: " ["PLATFORM NO. 62566", 57.7]

ex4/q2/q2.output.txt

4.3 Question 3

Le code pour cette question se trouve au chemin suivant :

ex4/q3/q3.py

Dans la fonction **mapper**, chaque ligne du jeu de données est analysée individuellement. On extrait le code du pays, les dates de début et de fin d'enregistrement, ainsi que le nom de la station. Si le code du pays n'est pas vide, on émet des paires clé-valeur avec le code du pays comme clé et un tuple contenant le nom de la station, la date de début et la date de fin comme valeur. La fonction **reducer** reçoit ces tuples groupés par pays. Elle itère ensuite sur ces enregistrements et calcule la différence entre les dates de début et de fin pour chaque station. La station ayant la plus grande différence de dates est identifiée. En cas d'égalité, le programme sélectionne la station avec le nom le plus long. **Les résultats** sont émis avec le code du pays en tant que clé et le nom de la station correspondante.

Les valeurs de sortie de ce programme se trouvent au chemin suivant :

ex4/q3/q3.output.txt

4.4 Question 4

La conversion individuelle des températures TMIN et TMAX dans la phase de mappage pose problème, car les paires (tmin, tmax) peuvent ne pas provenir du même enregistrement (par exemple, de dates différentes) lors de la phase de réduction. De plus, nous ne récupérons pas les enregistrements associés à la station Central Park seulement mais pour tous les stations.

4.5 Question 5

Le code pour cette question se trouve aux chemins suivants :

ex4/q5/q5.py (output=une seule colonne csv : diff tmp)
ex4/q5/q5_with_date.py (output=2 colonnes csv : date and diff tmp)

Les clés émises par **le mapper** sont les dates, et les valeurs sont des tuples composés du type de température et de la valeur numérique de la température. Les types de valeurs émises par le mapper sont de la forme (type, valeur), où le type peut être 'TMAX' ou 'TMIN', et la valeur est un entier représentant la température.

(clé=date, valeur=(type, valeur temp))

Afin d'avoir une seule colonne en sortie du programme, le protocole

de sortie JSONValueProtocol a été utilisé. Pour un meilleur output, le CsvProtocol de mr3px.csvprotocol peut aussi être utilisé.

Les valeurs de sortie de ce programme se trouvent aux chemins suivants :

ex4/q5/q5.output.csv
ex4/q5/q5_with_date.output.csv

4.6 Question 6

Le mapper ne peut pas produire une paire (clé,valeur) depuis une seule ligne du dataset car les informations TMIN et TMAX pour une date ne sont pas présentes sur la même ligne dans le dataset. Par exemple:

- USW00094728,20130101,**TMAX**,44,X,2400
- USW00094728,20130101,**TMIN**,-33,X,2400

La solution est de regrouper les valuers TMIN ET TMAX en utlisant la date comme clé dans le mapper
Si l'on prend l'exemple ci-dessus : après le mappage, nous aurons les données sous la forme suivante **(clé=date, valeur=(type, valeur_temp)**) :
(20130101, TMAX, 44)
(20130101, TMIN, -33)
Après l'étape de shuffle et de tri, les données seront réorganisées comme suit :
(20130101, [(TMAX, 44), (TMIN, -33)])

4.7 Question 7

Le reducer prend en compte les températures TMIN et TMAX associées à une même date, triées par type, et calcule la différence de température entre la maximale et la minimale en divisant par 10 pour la conversion en degrés Celsius. Cela assure une correspondance précise des températures pour chaque mesure. Par exemple (20130101, [(TMAX, 44), (TMIN, -33)]) → 7.7

Voici une ref dans le output actuel du program :
Q5 Justif2

4.8 Question 8

Veuillez consulter les images montrant l'exécution avec Hadoop sur la page annexe avec le runtime.

Q5 Hadoop Exec
Q5 Hadoop Application Runtime :1 step in 39s

4.9 Question 9

Le code pour cette question se trouve aux chemins suivants :

ex4/q9/q9.py (output=une seule colonne csv : avg tmp)
ex4/q9/q9_with_station_name.py (output= 2 colonnes csv: station name et avg tmp)

4.9.1 Le Mapper

- Le mapper prend chaque ligne du fichier d'entrée (qui est un fichier CSV) et extrait les informations nécessaires sur la station, la date et la température.
- Si la ligne contient suffisamment d'informations et la mesure est soit 'TMAX' ou 'TMIN', le mapper émet une paire clé-valeur avec la station comme clé et un tuple contenant **la date, le type de température et la valeur**.

4.9.2 Le combiner

- Le combiner effectue une agrégation locale des données du mapper en combinant les informations de température pour chaque date.
- Il utilise un dictionnaire (date_temp_dict) pour stocker les informations de température pour chaque date.
- La clé de sortie est la station, et la valeur est une liste de tuples contenant la date et une liste des informations de température correspondantes.

4.9.3 Reducer - Première étape

- Combine les dictionnaires générés par le combiner.
- Convertit les dictionnaires en une liste de tuples (date, temp_list).
- Calcule la différence de température entre 'TMAX' et 'TMIN' pour chaque date et émet une paire clé-valeur avec la station comme clé et une liste de tuples contenant **la date et la différence de température**.

4.9.4 Reducer - Deuxième étape

- Aplatit la liste de listes générée par la première étape.
- Calcule la moyenne des différences de température pour chaque station sur toutes les dates disponibles.
- Émet une paire clé-valeur avec la station comme clé et **la moyenne des différences de température comme valeur**.

4.9.5 Mrjob Steps

La classe MrJob utilise la bibliothèque MrJob et déclare deux étapes :

- La première étape utilise le mapper, le combiner et le reducer de la première étape.
- La deuxième étape utilise seulement le reducer de la deuxième étape.

4.9.6 Exemple de sortie post-combiner

Pour la station "AE000041196", la liste contient des tuples de la forme (**"date", [["TMAX", valeur], ["TMIN", valeur]]). Q9 Post Combiner Output Step**

4.9.7 Exemple de sortie post-première réduction

Pour la station "E00004119", la liste contient des tuples de la forme **("date", différence de température).Q9 Post First Reducer Step**

4.9.8 Exemple de sortie finale post-deuxième réduction

La sortie finale consiste en une paire clé-valeur où la clé est None et la valeur est la moyenne des différences de température pour chaque station. Afin d'avoir une seule colonne en sortie du programme, le protocole de sortie JSONValueProtocol a été utilisé.**Q9 Final Output with one csv column : average temp diff of each station**

Q9 Hadoop Exec

Les valeurs de sortie de ce programme se trouvent aux chemins suivant :

ex4/q9/q9.output.csv
ex4/q9/ex4/q9/q9_with_station_name_output.csv

Pour les étapes intermédiaires, veuillez consulter les chemins suivants. Le fichier du combineur est volumineux, il faut le dézipper.

ex4/q9/steps/q9_post_combiner_output.csv.gz
ex4/q9/steps/q9_post_first_reducer_output.csv

A Annexes

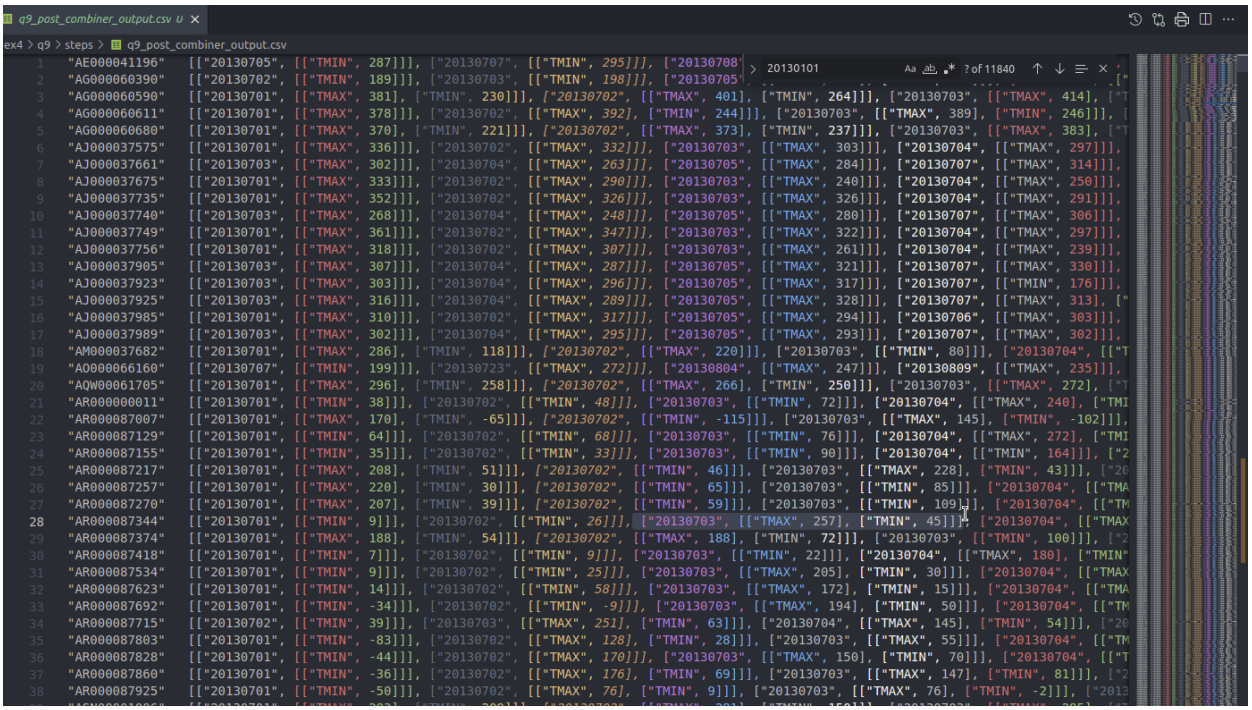


Figure 1: Q9 Post Combiner Output Step

Figure 2: Q9 Post First Reducer Step

Figure 3: Q9 Final Output with one csv column : average temp diff of each station

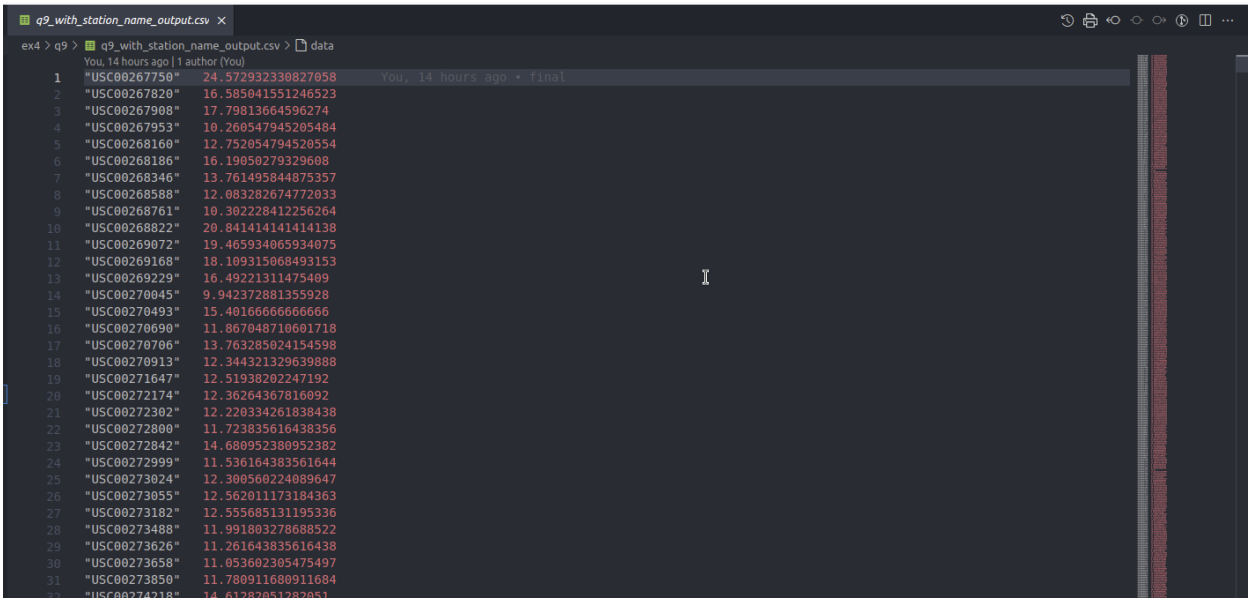


Figure 4: Q9 Final Output with station name: average temp diff of each station

```
q1/ q2/ q3/ q3/ q9/
ahmad@ahmad-pc:~$ python3 bd-analytics/tp1-bd-analytics/ex4/q9/q9.py -r hadoop
hdfs:///user/ahmad/ncdc-2013-sorted.csv
No configs found; falling back on auto-configuration
No configs specified for hadoop runner
Looking for hadoop binary in /home/ahmad/hadoop/bin...
Found hadoop binary: /home/ahmad/hadoop/bin/hadoop
Using Hadoop version 3.3.6
Looking for Hadoop streaming jar in /home/ahmad/hadoop/...
Found Hadoop streaming jar: /home/ahmad/hadoop/share/hadoop/tools/lib/hadoop-st
reaming-3.3.6.jar
Creating temp directory /tmp/q9.ahmad.20240309.120133.683991
uploading working dir files to hdfs:///user/ahmad/tmp/mrjob/q9.ahmad.20240309.1
20133.683991/files/wd...
Copying other local files to hdfs:///user/ahmad/tmp/mrjob/q9.ahmad.20240309.120
133.683991/files/
Running step 1 of 2
```

Figure 5: Q9 Hadoop Exec

Capacity Scheduler [memory-mb (unit=Mi), vcores] <memory:1024, vCores:1> <memory:8192, vCo

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	Final
application_1709985432781_0002	ahmad	streamjob1702927450958000783 4.jar	MAPREDUCE		default	0	Sat Mar 9 12:02:46 +0000 2024	Sat Mar 9 12:02:49 +0000 2024	Sat Mar 9 12:03:03 +0000 2024	FINISHED	SUCCE
application_1709985432781_0001	ahmad	streamjob3905373136913480159.jar	MAPREDUCE		default	0	Sat Mar 9 12:01:43 +0000 2024	Sat Mar 9 12:01:44 +0000 2024	Sat Mar 9 12:02:42 +0000 2024	FINISHED	SUCCE

Showing 1 to 2 of 2 entries

Figure 6: Q9 Hadoop Application Runtime : 2 steps in 1min 20s

```
USW00094728,20130101,TMAX,44,,,X,2400
USW00094728,20130101,TMIN,-33,,,X,2400
```

Figure 7: Q5 Justif1

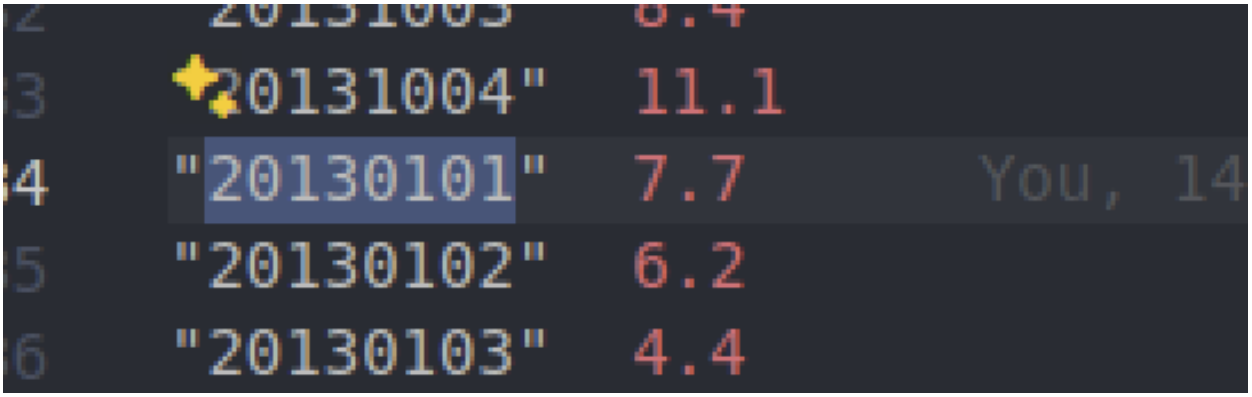


Figure 8: Q5 Justif2

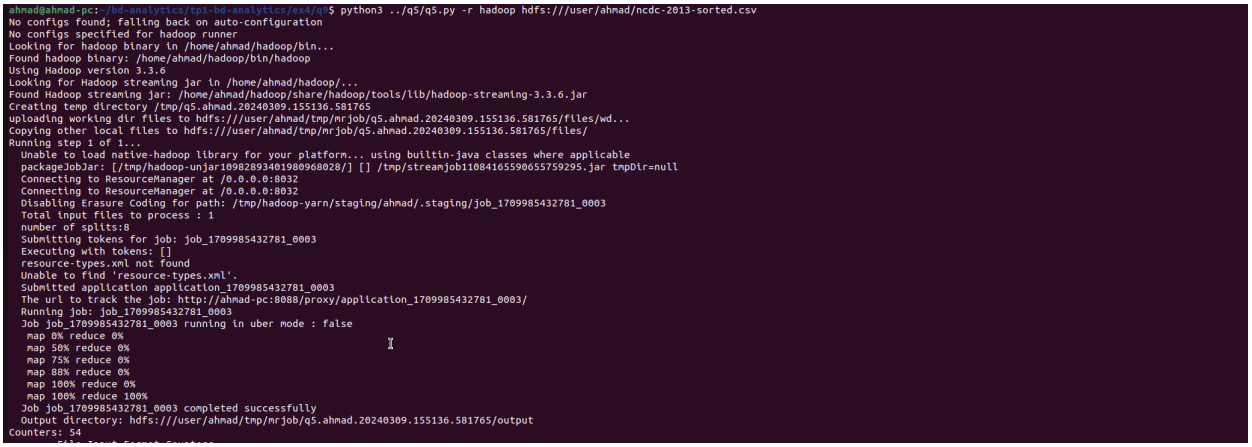


Figure 9: Q5 Hadoop Exec

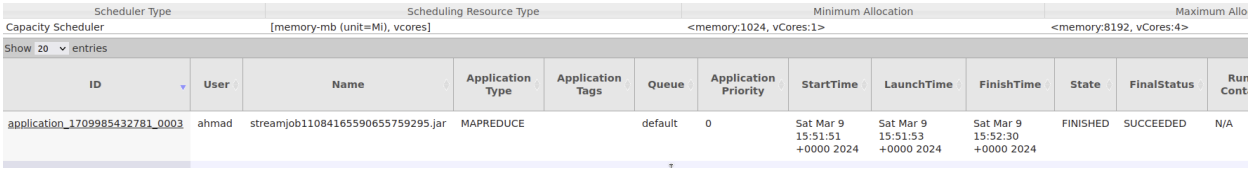


Figure 10: Q5 Hadoop Application Runtime :1 step in 39s