Τεχνολογίες Αποκεντρωμένων Δεδομένων

Project Distributed Hash Tables - Project Spark 2024

Αγγελόπουλος Γεώργιος - ΑΜ: 1067435
Δελάλης Βασίλειος - ΑΜ: 1067376
Σοϊλεμεζίδης Αλέξανδρος - Παναγιώτης - ΑΜ: 1067522

Web Crawler

Παραδοχες:
Σκοπός ήταν να κατασκευαστεί ένα *web crawler* γραμμένο στην python το οποίο θα έπαιρνε τα δεδομένα από ένα σύνδεσμο της wikipedia με όλους τους επιστήμονες της επιστήμης των υπολογιστών. Όμως υπήρχε το θέμα ότι στο αρχικό λινκ που δόθηκε υπήρχαν μόνο τα ονόματα των επιστημόνων και όχι πιο προσωπικές πληροφορίες τους που ζητούσε η εκφώνηση. Έτσι, για να πάρουμε όσα χρειαζόντουσαν, χρειάστηκε να επισκεφθούν όλες οι σελίδες της wikipedia των επιστ
μών.

Επίσης κάθε επιστήμονα η σελίδα είχε πολλές φορές διαφορετική διαρρυθμιση και οι πληροφορίες ήταν σε πολλές περιπτώσεις δυσπρόσιτες, οπότε κάναμε μια παραδοχή ότι θα λαμβάναμε πληροφορίες μόνο από τον πίνακα πληροφοριών του κάθε επιστήμονα, αν αυτός υπήρχε. Εκεί ψάχναμε για τα Alma Mater και Awards πεδία και όπου αυτά υπήρχαν παίρναμε όλα τα δεδομένα τους και τα αποθηκεύαμε.

Για να επιτευχθεί όλο αυτό υλοποιήθηκαν 2 βασικές συναρτήσεις (**Update_Scientists_Record()**, **Update_Final_Record()**) στο python αρχείο και μία βοηθητική για λόγους δοκιμών και ευκολίας χρήσης (**test()**).
Επίσης χρησιμοποιήθηκαν μερικές βοηθητικές βιβλιοθήκες για την σωστή λειτουργία του crawler (bs4, requests, csv, re, json)

```
from bs4 import BeautifulSoup  # dependancies
import requests  # dependancies
import csv
import re
import json
```

Short description of code implementation/τρένο σκεψης:

Όπως αναφέρθηκε, υπάρχουν δύο συναρτήσεις που υλοποιούν το crawler στην ολότητα του.
1) Η πρώτη στοχεύει στην προσκόμηση όλων των επιστημόνων από την λίστα της wikipedia και των συνδέσμων προς τις προσωπικες τους σελίδες στον ίδιο ιστότοπο. Αρχικά χρησιμοποιούμε το λινκ από την εκφώνηση για να φορτώσουμε την σελίδα στο beautifulsoup και να αρχίσουμε να κοιτάμε τα στοιχεία του καθώς και ορίζουμε μία λίστα για προσωρινή αποθήκευση των δεδομένων μας.

```python
def Update_Scientists_Record():
    temp_table_of_scientists = []
    letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

    # Making the soup.
    url = "https://en.wikipedia.org/wiki/List_of_computer_scientists"
    result = requests.get(url)
    doc = BeautifulSoup(result.text, "html.parser")

```

Έπειτα, μετά από παρατήρηση της δομής της σελίδας παρατηρήσαμε ότι η παράθεση των ονομάτων είναι σε μορφή λίστας και διακεκριμένα κατά βάση επωνύμων των επιστημόνων. Στην html δόμηση αυτό είχε h2 tags με κείμενο το κάθε γράμμα της αλφαβήτου που κάθε επώνυμο ξεκίναγε. Άρα απλά υλοποιήθηκε ένα for loop όπου κάθε iteration πιάνει και ενα διαφορετικό γράμμα και αυτο χρησιμοποιείται για να κάνουμε scope in στο αντίστοιχο σημειο της λίστας. Όταν βρεθεί αυτό το σημείο, αποθηκεύουμε στο *current_ul* όλα τα *li* στοιχεία που τυχαίνει να είναι οι επιστήμονες.

```python
for letter in letters:
    try:
        tags = doc.find(class_="mw-content-ltr mw-parser-output").find(
            class_="mw-headline", id=letter
        )
        current_ul = tags.parent.next_sibling.next_sibling.find_all("li")
```

Τότε, για κάθε στοιχείο στο current_ul, πάμε και αποθηκεύουμε στην προσωρινή λίστα μας το όνομα και τον σύνδεσμο του κάθε επιστήμονα σαν μία λίστα 2 στοιχείων.

Αν από πριν δεν βρεθεί κάποιο h2 tag με το γράμμα που ζητείται, σημαίνει ότι απλά δεν υπάρχουν επιστήμονες με επώνυμο που ξεκινάει από αυτό το γράμμα οπότε το προσπερνάμε και πάμε στο επόμενο γράμμα.

```python

            # Append the temporal list of scientists with the links to their wikipedia
            for li in current_ul:
                temp_table_of_scientists.append(
                    [li.a["title"], "https://en.wikipedia.org" + li.a["href"]]
                )

        except:
            print("There's no scientist in the letter: " + letter)
```

Στο τέλος της λούπας, στην προσωρινή λίστα μας υπάρχουν όλοι οι επιστήμονες και τα λινκ τους. Τότε καλούμε την csv βιβλιοθήκη για να τα αποθηκεύσουμε για μετέπειτα χρήση χωρίς την ανάγκη να ξανατρέξουμε αυτό το κομμάτι κώδικα μέχρι να χρειαστεί κάποιο update.

```python
    # create the csv for the scientists and the links.
    with open(smallTempCSV, "w", encoding="utf-8", newline="") as csvfile:
        csvwriter = csv.writer(csvfile)
        csvwriter.writerow(["Scientist", "Link"])
        csvwriter.writerows(temp_table_of_scientists)

    print("Update completed!")
```

2) Η δεύτερη συνάρτηση μας έχει σκοπό να βρει τα βραβεία και τα ιδρύματα που αποφοίτησαν οι επιστήμονες. Με βάση τις παραδοχές που κάναμε, αυτα τα βρίσκουμε στα αντίστοιχα πεδία αν υπάρχουν στον πίνακα βιογραφίας του κάθε επιστήμονα. Με αυτά κατα νου, η συνάρτηση δουλεύει ως εξης:
Αρχικά δημιουργούμε μερικές βοηθητικές λίστες για αποθήκευση προσωρινών δεδομένων. Έπειτα δημιουργούμε μία λίστα με τα regular expressions που απευθύνονται στο Alma mater διότι παρατηρήθηκε ότι μερικοι επιστήμονες δεν είχαν πεδίο Alma mater αλλα education.

```
52  # function that parses the scientists CSV and opens all the links fetching inf
53  def Update_Final_Record():
54      list_of_dictionaries = (
55          []
56      )  # list that will hold the dictionaries needed for the json file
57      has_no_table = (
58          []
59      )  # list that holds the names and links of scientists with no info table
60      has_no_awards = (
61          []
62      )  # list that holds the names and links of scientists with no awards  in
63      has_no_alma = (
64          []
65      )  ##list that holds the names and links of scientists with no Alma mater
66
67      alma_patterns = ["Alma*", "Education*"]
68      alma_patterns = [re.compile(p) for p in alma_patterns]
```

Στην συνέχεια, ανοίγουμε το αρχείο που δημιουργήσαμε πριν με τους επιστήμονες και τους συνδέσμους τους και αποθηκεύουμε σε μία λίστα τα ονόματα και τα λινκ τους.

```
69
70      with open(
71          smallTempCSV, "r", encoding="utf-8", newline=""
72      ) as names_links:  # Read from the csv of the scientists
73          scientists_links = csv.reader(names_links, delimiter=",")
74          next(scientists_links, None)  # Skip header
```

Δημιουργούμε ένα for loop για κάθε επιστήμονα στην λίστα και δημιουργούμε δύο λίστες, μία για awards και μια για alma mater. Επισης χρησιμοποιούμε σε κάθε loop το αντίστοιχο λινκ του κάθε επιστήμονα και φορτώνουμε την σελίδα του στο beautifulsoup.

```
75          # For every scientist in the csv, visit the link of the
76          # Each time it creates a new soup tree.
77          for name, url in scientists_links:
78              awards = []
79              alma_mater = []
80
81              result = requests.get(url)
82              doc = BeautifulSoup(result.text, "html.parser")
83
```

Αρχικά τσεκάρουμε για τον επιστήμονα αν υπαρχει στην σελίδα του infobox, πίνακας πληροφοριών. Αν όχι δεν ασχολούμαστε με αυτόν τον επιστήμονα και τον αποθηκεύουμε στην λιστα με επιστήμονες χωρίς πληροφορία. Αν έχει όμως, αποθηκεύουμε στην μεταβλητή pos την θέση μέσα στον html tree του infobox.

```
84          if doc.find(class_="infobox"):
85              # specify the desired position in the soup tr
86              pos = doc.find(class_="infobox").tbody
87
88              # handle unexpected errors as exceptions
89  >           try:…
119
120 >           except:…
122
123         else:
124             has_no_table.append([name, url])
125             print(name + " has no info table")
126
```

Στην συνέχεια, αν έχει infobox, ανοίγουμε ένα try section για την διαχείριση τυχόν σφαλμάτων χωρίς την διακοπή της λειτουργίας του προγράμματος καθώς και την τύπωση αντίστοιχου κειμένου για να γνωρίζουμε σε ποιον επιστήμονα προέκυψε το λάθος.

```
88              # handle unexpected errors as exceptions
89  >           try:…
119
120             except:
121                 print("With " + name + " occured an error!!")
```

Αρχικά ασχολούμαστε να βρούμε το Alma Mater του επιστήμονα. Ξεκινάμε ένα μικρό for loop που ψάχνουμε για το tag με όνομα alma mater ή education με βάση τον μικρό πίνακα που φτιάξαμε πριν για να βρουμε την πληροφορία που θέλουμε. Αν δεν βρεθεί τίποτα από τα δύο, βγαίνουμε από το for loop και προσθετουμε τον επιστήμονα στην λίστα με αυτούς που δεν εχουν alma mater τυπώνοντας αντίστοιχο κείμενο.
Παίρνουμε όλα τα “**a**” tags που βρίσκουμε και τα φιλτράρουμε ώστε να κρατήσουμε το string περιεχόμενο μόνο από αυτά που αναφέρονται σε ιδρύματα (αντι πχ χρονολογιες).
Τέλος αποθηκεύουμε κάθε string στον πίνακα **alma_mater**.

```
90          # Alma mater
91          for pattern in alma_patterns:
92              alma_section = pos.find("th", string=pattern)
93              if alma_section is None:
94                  continue
95
96              tags = alma_section.next_sibling.find_all("a")
97              for tag in tags:
98                  # Filter out Wikipedia tags, like [citation needed]
99                  if not any(parent.name == "sup" for parent in tag.parents):
100                     alma_mater.append(tag.string)
101             break
102         else:
103             has_no_alma.append([name, url])
104             print(name + " has no alma mater record (skill issue :C )")
```

Ομοίως κινούμαστε για τα Awards. Ψάχνουμε να βρούμε αν υπάρχει το πεδίο Awards για τον επιστήμονα. Αν υπάρχει, παίρνουμε όλα τα "**a**" tags του και τα φιλτράρουμε για να βρούμε τα string όλων των awards τα οποία στην συνέχεια τα αποθηκεύουμε στον πινακα **awards.**
Αν δεν υπάρχει, προσθέτουμε το όνομα του και το λινκ του στην λίστα με τους επιστήμονες χωρις πεδίο awards και τυπώνουμε αντίστοιχο κείμενο.

Στο τέλος όλων αυτών τυπώνουμε μήνυμα ότι η προσκόμιση πληροφορίας για τον επιστήμονα τελείωσε.

```
106         # Awards
107         awards_section = pos.find("th", string="Awards")
108         if awards_section is not None:
109             tags = awards_section.next_sibling.find_all("a")
110             for tag in tags:
111                 if not any(parent.name == "sup" for parent in tag.parents):
112                     awards.append(tag.string)
113         else:
114             has_no_awards.append([name, url])
115             print(name + " has no award record (cry about it :C )")
116 
117         # success check for the current scientist
118         print("Done with " + name)
```

Στην συνέχεια, για τον ίδιο επιστήμονα, δημιουργούμε ένα προσωρινό dictionary με το όνομα και τις λιστες alma mater, awards, ακόμα και αν είναι κενές μετά τα προηγούμενα. Τέλος, αποθηκεύουμε αυτό το dictionary στην λίστα με τα dictionaries, η οποία περιέχει τα dictionaries όλων των επιστημόνων.

```
126 
127         # Create a temporal dictionary to add it to the list for saving purposes
128         temp_dict = {
129             "name": name,
130             "alma_mater": alma_mater,
131             "awards": awards,
132         }
133         # Add the fetched info to the list as a dictionary for easier cenverting
134         list_of_dictionaries.append(temp_dict)
135 
```

Μόλις τελειώσει το for loop και έχει περάσει από κάθε επιστήμονα στο csv αρχείο, δημιουργουμε τρία αρχεία, ένα json που είναι το ζητούμενο και δύο csv για στατιστική αποθήκευση επιστημόνων που δεν είχαν awards ή alma mater πεδία.
Στο JSON αρχείο αποθηκεύουμε την λίστα με τα dictionaries, όπου από πριν είναι έτοιμα μορφοποιημένη η πληροφορία που ζητάμε.

```python
    with open("no_awards.csv", "w", encoding="utf-8", newline="") as noAwards:
        writer = csv.writer(noAwards)
        writer.writerow(["Scientist", "Link"])
        writer.writerows(has_no_awards)

    with open("no_alma_mater.csv", "w", encoding="utf-8", newline="") as noAlma:
        writer = csv.writer(noAlma)
        writer.writerow(["Scientist", "Link"])
        writer.writerows(has_no_alma)

    with open(Final_Record, "w", encoding="utf-8") as outfile:
        json.dump(list_of_dictionaries, outfile, indent=4, ensure_ascii=False)

```

Sample output:

Παρακάτω βρίσκονται εικόνες παραδειγμάτων τυπωσεων μηνυμάτων που αναφέρθηκαν πριν καθώς και μερικά πεδία από τα αναφερόμενα αρχεία αρχεία.

Ενημέρωση του πίνακα με τους επιστήμονες και τους συνδέσμους τους, το αρχείο που παράγεται:

```
"
1. Update the list of scientists.
2. Update info of scientists.
3. Quit.

1
There's no scientist in the letter: Q
There's no scientist in the letter: X
Update completed!
```

	Scientist	Link
2	Atta ur Rehman Khan	https://en.wikipedia.org/wiki/Atta_ur_Rehman_Khan
3	Wil van der Aalst	https://en.wikipedia.org/wiki/Wil_van_der_Aalst
4	Scott Aaronson	https://en.wikipedia.org/wiki/Scott_Aaronson
5	Rediet Abebe	https://en.wikipedia.org/wiki/Rediet_Abebe
6	Hal Abelson	https://en.wikipedia.org/wiki/Hal_Abelson
7	Serge Abiteboul	https://en.wikipedia.org/wiki/Serge_Abiteboul
8	Samson Abramsky	https://en.wikipedia.org/wiki/Samson_Abramsky
9	Leonard Adleman	https://en.wikipedia.org/wiki/Leonard_Adleman
10	Manindra Agrawal	https://en.wikipedia.org/wiki/Manindra_Agrawal
11	Luis von Ahn	https://en.wikipedia.org/wiki/Luis_von_Ahn
12	Alfred Aho	https://en.wikipedia.org/wiki/Alfred_Aho
13	Frances E. Allen	https://en.wikipedia.org/wiki/Frances_E._Allen
14	Gene Amdahl	https://en.wikipedia.org/wiki/Gene_Amdahl
15	David P. Anderson	https://en.wikipedia.org/wiki/David_P._Anderson
16	Lisa Anthony	https://en.wikipedia.org/wiki/Lisa_Anthony
17	Andrew Appel	https://en.wikipedia.org/wiki/Andrew_Appel
18	Cecilia R. Aragon	https://en.wikipedia.org/wiki/Cecilia_R._Aragon
19	Bruce Arden	https://en.wikipedia.org/wiki/Bruce_Arden
20	Angie Jones	https://en.wikipedia.org/wiki/Angie_Jones
21	Sanjeev Arora	https://en.wikipedia.org/wiki/Sanjeev_Arora
22	Winifred Asprey	https://en.wikipedia.org/wiki/Winifred_Asprey

Ενημέρωση του τελικού αρχείου JSON με τις πληροφορίες που ζητούνται καθώς και κομμάτι του αρχείου αυτού:

```
Aaron Sloman has no award record (cry about it :C )
Done with Aaron Sloman
Arne Sølvberg has no info table
Brian Cantwell Smith has no award record (cry about it :C )
Done with Brian Cantwell Smith
Done with David Canfield Smith
Steven Spewak has no info table
Carol Spradling has no award record (cry about it :C )
Done with Carol Spradling
Robert Sproull has no award record (cry about it :C )
Done with Robert Sproull
Rohini Kesavan Srihari has no award record (cry about it :C )
Done with Rohini Kesavan Srihari
Done with Sargur Srihari
Maciej Stachowiak has no award record (cry about it :C )
Done with Maciej Stachowiak
Done with Richard Stallman
Ronald Stamper has no info table
Thad Starner has no alma mater record (skill issue :C )
Thad Starner has no award record (cry about it :C )
Done with Thad Starner
Done with Richard E. Stearns
Done with Guy L. Steele, Jr.
Done with Thomas Sterling (computing)
Alexander Stepanov has no info table
W. Richard Stevens has no award record (cry about it :C )
Done with W. Richard Stevens
Larry Stockmeyer has no alma mater record (skill issue :C )
Larry Stockmeyer has no award record (cry about it :C )
Done with Larry Stockmeyer
Salvatore Stolfo has no info table
Done with Michael Stonebraker
Olaf Storaasli (computing) has no alma mater record (skill issue :C )
Olaf Storaasli (computing) has no award record (cry about it :C )
Done with Olaf Storaasli (computing)
Christopher Strachey has no award record (cry about it :C )
Done with Christopher Strachey
Volker Strassen has no award record (cry about it :C )
```

Final_Record.json


```
[
    {
        "scientist": "Atta ur Rehman Khan",
        "Alma mater": [
            "Technical University of Berlin"
        ],
        "Awards": [
            "Werner von Siemens Ring",
            "Harry H. Goode Memorial Award",
            "George Stibitz",
            "Wilhelm Exner Medal",
            "Order of Merit of the Federal Republic of Germany",
            "Computer History Museum"
        ]
    },
    {
        "scientist": "Wil van der Aalst",
        "Alma mater": [
            "Technical University of Berlin"
        ],
        "Awards": [
            "Werner von Siemens Ring",
            "Harry H. Goode Memorial Award",
            "George Stibitz",
            "Wilhelm Exner Medal",
            "Order of Merit of the Federal Republic of Germany",
            "Computer History Museum"
        ]
    },
```

Υποσημείωση:

1) Μερικές συντακτικές επιλογές του beautiful soup εγιναν για να γίνει σωστή ανάγνωση του αρχείου και του δέντρου που κατασκευάζει το beautiful soup.
2) Επίσης, σε πολλές περιπτώσεις οι επιστήμονες φαίνονται να μην έχουν βραβεία ή ιδρύματα αποφοίτησης. Αυτό οφείλεται στην απουσία πεδίων που να αναφέρουν αυτά τα στοιχεία ή σε πολύ ακραίες περιπτώσεις διαφορετικής διατύπωσης τους στον html κώδικα. Δεν μπορούμε να περάσουμε κάθε σελίδα ξεχωριστά και να προσαρμόσουμε τον κώδικα στα μέτρα της, χάνεται ή ουσία της αυτοματοποίησης από ένα bot. Όμως έχει καλυφθεί ένα τεράστιο μέρος των πληροφοριών που μας δίνονται και το πρόγραμμα δουλεύει σε μεγάλο ποσοστό.
3) Υπάρχουν περιπτώσεις που μαζί με το λινκ για ενα πανεπιστήμιο, σε κάποιου επιστήμονα το πεδίο γίνεται αναφορά και στην εργασία του με παραπομπές σε αυτήν με υπερσυνδέσμους. Επειδή το crawler πιάνει όλα τα a tags στο ίδιο επίπεδο, τα βρίσκει ως ίδια με αυτά των ιδρυμάτων και τα προσθέτει μαζί τους. Δυστυχώς δεν μπορούμε να βρούμε εύκολη λύση σε αυτό το πρόβλημα αλλα προκαλεί πολύ μικρό θέμα.

Chord DHT

Για αυτό το κομμάτι της εργασίας υλοποιήσαμε ένα DHT (distributed hash table/κατανεμημένος πίνακας κατακερματισμού) με τη χρήση του πρωτοκόλλου Chord. Το Chord δημιουργήθηκε το 2001 από μια ομάδα ερευνητών στο MIT, και αποτελεί ένα από τα πρώτα και πιο γνωστά πρωτόκολλα για DHT. Αν και δε χρησιμοποιείται πια σε πραγματικά συστήματα, έχει ακόμα διδακτική αξία λόγω της απλότητάς του.

Πρωτόκολλο

Μία από τα βασικές ιδιότητες των DHT είναι η δυνατότητα κάθε κόμβου να βρει σε ποιον κόμβο βρίσκεται ένα δεδομένο κλειδί. Για να εκπληρώσει αυτό το ζητούμενο, το Chord χρησιμοποιεί μια τεχνική γνωστή ως consistent hashing, στην οποία ορίζουμε έναν δακτύλιο και αντιστοιχίζουμε κάθε κόμβο και κάθε πιθανό κλειδί με ένα σημείο πάνω σε αυτόν, το οποίο αποκαλούμε το *id* του. Αφού τα κλειδιά και οι κόμβοι μοιράζονται αυτόν τον χώρο, μπορούμε να αντιστοιχίσουμε ντετερμινιστικά και χωρίς περαιτέρω πληροφορίες ένα δεδομένο κλειδί με έναν κόμβο. Άρα, μπορούμε να υπολογίσουμε σε ποιον κόμβο θα πρέπει να βρίσκεται ένα κλειδί χωρίς να είναι απαραίτητη η επικοινωνία ανάμεσα στους κόμβους. Για την αντιστοίχιση κόμβων και κλειδιών με τα id τους, χρησιμοποιείται μια συνάρτηση κατακερματισμού της οποίας η έξοδος ερμηνεύεται ως θετικός αριθμός.

Το Chord δημιουργεί έναν δακτύλιο κόμβων που βασίζεται στον δακτύλιο του consistent hashing. Οι κόμβοι είναι αρχικά μεμονωμένοι και κατασκευάζουν σταδιακά τον δακτύλιο επικοινωνώντας μεταξύ τους. Συνεχίζουν να ελέγχουν την ακεραιότητα του δακτυλίου επ' άπειρον (λειτουργία stabilize()) και την επαναφέρουν μετά από εισόδους και εξόδους κόμβων, σφάλματα στο δίκτυο, κλπ. Καθώς η διαδικασία αυτή δεν είναι άμεση, το σύστημα είναι εντελώς ακέραιο μόνο στη σταθερή κατάσταση.

Βασική έννοια του πρωτοκόλλου αποτελεί η ιδέα του *διαδόχου* ενός id, που είναι ο πρώτος κόμβος που συναντούμε αν κινηθούμε δεξιόστροφα με αρχή το id. Ο διάδοχος ενός κλειδιού είναι ο κόμβος που το περιέχει, ενώ ο διάδοχος ενός κόμβου είναι ο κύριος «παρτενέρ» του όσον αφορά την επικοινωνία. Ο δακτύλιος του Chord είναι μια κλειστή αλυσίδα από κόμβους, ο καθένας διάδοχος του προηγούμενου.

Για να μπορεί ένας κόμβος να επικοινωνήσει με τον κόμβο αποθήκευσης ενός κλειδιού, θα πρέπει να μπορει να τον βρει είτε άμεσα είτε έμμεσα. Για να έχουν όλοι οι κόμβοι άμεση επικοινωνία μεταξύ τους, πρέπει να μοιράζονται συνεχώς τους κόμβους που ξέρουν (gossip protocol). Αν αντιθέτως οι κόμβοι μπορούσαν να μιλήσουν μόνο με τους διαδόχους τους, θα χρειαζόταν γραμμικός χρόνος μέχρι να φτάσει το μήνυμα στον στόχο. Το Chord επιλέγει μια ενδιάμεση στρατηγική, χρησιμοποιώντας έναν πίνακα με «δακτύλους».

Κάθε κόμβος x διατηρεί έναν πίνακα με κόμβους που βρίσκονται δεξιόστροφα από τον ίδιο, τους οποίους αποκαλούμε δακτύλους. Ο πρώτος δάκτυλος είναι ο διάδοχος. Ο i-οστός είναι ο διάδοχος του id(x) + 2^i, με μέγιστο i το μέγεθος της εξόδου της συνάρτησης κατακερματισμού σε bits. Αυξάνοντας γεωμετρικά την απόσταση ανάμεσα σε διαδοχικούς δακτύλους, καλύπτουμε το μεγαλύτερο μέρος του δακτυλίου και ταυτόχρονα έχουμε μεγάλη ακρίβεια στις κοντινές αναζητήσεις. Με αυτή την τεχνική, η αναζήτηση έχει πολυπλοκότητα O(log N) w.h.p.

Καθώς οι δάκτυλοι είναι δείκτες σε κόμβους του δακτυλίου, χρειάζονται κι αυτοί συντήρηση. Αυτό γίνεται με τη λειτουργία fix_finger(), η οποία καλείται περιοδικά (όπως και η stabilize()) και στέλνει τα απαραίτητα μηνύματα για να ελέγξει αν η τιμή ενός δακτύλου πρέπει να αλλάξει.

Υλοποίηση

Δημιουργήσαμε δύο υλοποιήσεις – μία κύρια και μία εναλλακτική – του ζητούμενου DHT με χρήση της Python και κάποιων ενσωματωμένων βιβλιοθηκών. Και στις δύο ορίζουμε την κλάση Node, που υλοποιεί τις διάφορες λειτουργίες του κόμβου, και μια κλάση (Transport/Simulator) που αποτελεί αφηρημένη αναπαράσταση του δικτύου, επιτρέποντας την επικοινωνία από κόμβο σε κόμβο.

Στην εναλλακτική υλοποίηση, ένα Node καλεί τις συναρτήσεις κάποιου άλλου (έμμεσα, μέσω του Transport) για να επικοινωνήσει με αυτό. Αυτή είναι μια απλή προσέγγιση που όμως θέτει άνω όριο στον αριθμό των κόμβων, καθώς με αρκετά μεγάλο πλήθος δημιουργούνται call stacks που υπερβαίνουν το όριο βάθους αναδρομής της Python.

Στην κύρια υλοποίηση, η κύρια συνάρτηση ενός κόμβου είναι η receive(), με την οποία ο κόμβος λαμβάνει ένα μήνυμα και (προαιρετικά) επιστρέφει ένα δικό του. Το Simulator περιέχει μια ουρά προτεραιότητας με μηνύματα, και με τη συνάρτησή του process() αφαιρεί ένα ένα τα μηνύματα από την ουρά και τα στέλνει στον προορισμό τους. Αν ο προορισμός επιστρέψει ένα δικό του μήνυμα (π.χ. απάντηση σε ένα query), το προσθέτει στην ουρά ώστε να δρομολογηθεί.

Πέρα από την αποφυγή της αναδρομής, αυτή η προσέγγιση παρέχει τη δυνατότητα για αυτοματοποιημένο έλεγχο του συστήματος. Αν το κριτήριο ταξινόμησης της ουράς δεν είναι η σειρά εισαγωγής αλλά ένας χρόνος άφιξης, που μπορούμε να παράξουμε ντετερμινιστικά ή τυχαία, μπορούμε να προσομοιώσουμε με πολύ ρεαλιστικό τρόπο τις συνθήκες που επικρατούν στα πραγματικά δίκτυα, όπου τα μηνύματα μπορεί να έχουν αυθαίρετα μεγάλη καθυστέρηση στην άφιξη. Η συγκεκριμένη τεχνική προσομοίωσης χρησιμοποιείται στην ανάπτυξη πραγματικών κατανεμημένων συστημάτων, όπως το sync engine της Dropbox.

Εκτός από τις λειτουργίες περί πρωτοκόλλου, τα Nodes υλοποιούν και την εγγραφή και ανάγνωση δεδομένων από την τοπική τους «βάση δεδομένων», η οποία προσομοιώνεται με έναν πίνακα κατακερματισμού (O(1) πρόσβαση), του οποίου τα κλειδιά είναι εκπαιδευτικά ιδρύματα και οι τιμές είναι οι επιστήμονες που έχουν φοιτήσει εκεί. Το φιλτράρισμα με βάση τον αριθμό βραβείων στην ανάγνωση γίνεται με απλή γραμμική σάρωση, αλλά θα μπορούσε εύκολα να αναβαθμιστεί με κάποια πιο αποδοτική τεχνική (π.χ. δυαδική αναζήτηση).

Τέλος, η μετατροπή κόμβων και κλειδιών σε id γίνεται με την κλάση Id, η οποία εφαρμόζει τη συνάρτηση κατακερματισμού και υλοποιεί κάποιες μαθηματικές πράξεις (πρόσθεση, έλεγχος διαστήματος) απαραίτητες για το consistent hashing.

Αξιολόγηση

Αξιολογήσαμε μόνο την κύρια υλοποίηση του συστήματος. Η πειραματική διαδικασία είναι η εξής:

- Δημιουργούμε έναν νέο δακτύλιο με N κόμβους και εκτελούμε τις λειτουργίες συντήρησης (stabilize, fix_finger) M και K φορές αντιστοίχως. Οι αριθμοί N, M, και K δίνονται από τον χρήστη.
- Αρχίζοντας από έναν κόμβο και ακολουθώντας την αλυσίδα των διαδόχων, ελέγχουμε αν ο δακτύλιος είναι ακέραιος. Αν η αλυσίδα έχει λιγότερους από N κόμβους ή ο τελευταίος κόμβος δεν έχει ως διάδοχο τον πρώτο, τότε δεν καλέσαμε αρκετές φορές τη συνάρτηση stabilize() και κάποια αιτήματα θα αποτύχουν.
- Εκτελούμε έναν μεγάλο αριθμό αιτημάτων (lookups) με τυχαία κλειδιά σε τυχαίους κόμβους για να ελέγξουμε την ακεραιότητα και την ταχύτητα του δακτυλίου.
- Φορτώνουμε τα δεδομένα περί επιστημόνων από το αρχείο που παράγει το web crawler (βλ. προηγούμενη ενότητα), και τα γράφουμε στο DHT, επιλέγοντας έναν τυχαίο κόμβο ως διεπαφή/client.
- Δίνουμε τη δυνατότητα στον χρήστη να κάνει δικά του αιτήματα. Δίνοντας το όνομα ενός εκπαιδευτικού ιδρύματος, εμφανίζονται οι αντίστοιχοι επιστήμονες και ο αριθμός των βραβείων που έχει λάβει ο καθένας.

Επιλέγοντας διαφορετικές τιμές για τις παραμέτρους N, M, και K, είναι εμφανής η επίδραση των λειτουργιών συντήρησης στην ορθότητα και στην απόδοση του συστήματος.

Ακέραιος δακτύλιος				
N	**M**	**K**	**Χρόνος 1000 αιτημάτων (ms)**	**Ποσοστό επιτυχίας αιτημάτων**
256	256	128	46	100%
512	512	128	90	100%
1024	1024	128	185	100%

Ανεπαρκής σταθεροποίηση				
N	**M**	**K**	**Χρόνος 1000 αιτημάτων (ms)**	**Ποσοστό επιτυχίας αιτημάτων**

256	32	128	45	10% (104)
256	64	128	46	29% (288)
256	128	128	46	54% (536)

Ανεπαρκής βελτιστοποίηση				
N	**M**	**K**	**Χρόνος 1000 αιτημάτων (ms)**	**Ποσοστό επιτυχίας αιτημάτων**
256	256	16	1849	100%
256	256	32	1666	100%
256	256	64	1221	100%
256	256	96	700	100%

Παρατηρούμε ότι αν ο δακτύλιος δεν είναι ακέραιος, η πιθανότητα επιτυχίας ενός αιτήματος μειώνεται σημαντικά. Το ίδιο ισχύει για την απόδοση του συστήματος αν οι δάκτυλοι των κόμβων δεν είναι ενημερωμένοι.

```
Node count: 128
Stabilization passes: 128
Optimization passes: 128

Ring created in 0 ms
Stabilized in 78 ms
Optimized in 78 ms

Ring fully stable: True
Testing ring...
1000/1000 correct lookups (100%) in 29 ms
```

```
Loaded 687 scientists
Ignoring 164 scientists with no alma mater
Wrote 523 scientists to DHT in 266 transactions and 4 ms
```

```
Alma mater: Harvard University
Minimum award count: 2

Found 8 scientists on host node109 in 0 µs.
-----------------------------------------------
Award count | Name
-----------------------------------------------
          8 | Stephen Cook
          4 | Bill Gates
          3 | Dan Ingalls
          9 | Richard Karp
          6 | Butler Lampson
          6 | Marvin Minsky
          8 | Dennis Ritchie
          5 | Richard Stallman
```

Decentralized Query processing and Optimization in Apache Spark

Γενικές Παραδοχές

Το κομμάτι του Apache Spark υλοποιήθηκε σε τοπικό cluster το οποίο περιγράφεται παρακάτω. Επιλεξαμε να μη χρησιμοποιήσουμε την πλατφόρμα Datastacks διότι το δωρεάν Community Edition δεν μας επιτρέπει να χρησιμοποιήσουμε αυτούσια τα αρχεία δεδομένου που είναι απαραίτητα για την εκτέλεση των ζητούμενων, αλλά την φόρτωση τους σε proprietary Tables, τα οποία δεν υποστηρίζουν ορισμένους από τους τύπους αρχείων που απαιτούμε. Παρακάτω παραθέτουμε βασικές προδιαγραφές του υλικού και λογισμικού που χρησιμοποιούμε για την υλοποίηση και εκτέλεση των ζητούμενων queries.

Processor: 4 × Intel® Core™ i3-6006U CPU @ 2.00GHz
Memory: 7,6 GiB of RAM
System: Kubuntu 23.10 (Kernel Version: 6.5.0-17-generic (64-bit))
Python: 3.11.6
Spark/pyspark: 3.5.0
Findspark: 2.0.1

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Python version 3.11.6 (main, Oct  8 2023 05:06:43)
Spark context Web UI available at http://192.168.1.104:4040
Spark context available as 'sc' (master = local[*], app id = local-1708248281258).
SparkSession available as 'spark'.
>>>
```

Παρατηρήσεις:
Τα αρχεία κώδικα είναι στη γλώσσα python, και χρησιμοποιείται το API pyspark για την επικοινωνία με το σύστημα. Ο κώδικας χάριν ευκολίας εκτελείται μέσω Jupyter Notebook. Κάθε αρχείο περιλαμβάνει κώδικα για την φόρτωση και σωστή μορφοποίηση των δεδομένων εισόδου στο σύστημα, την εκτέλεση και των τεσσάρων ζητούμενων queries κάθε θέματος και την εύκολα αναγνώσιμη εκτύπωση των αποτελεσμάτων στο τερματικό.

Θέμα 1ο: Θερμοκρασία και Υγρασία

Αρχεία εισόδου, φόρτωση και μορφοποίηση

Τα αρχεία εισόδου (tempm.txt και hum.txt) είναι σε μορφή plain text, και περιέχουν πολλαπλά αρχεία τύπου json. Κάθε αρχείο json (κάθε γραμμή κειμένου) έχει δεδομένα χρόνου (timestamps) και αναλόγως το αρχείο εισόδου, δεδομένα θερμοκρασίας και υγρασίας αντίστοιχα για κάθε timestamp. Κάθε αρχείο json έχει δεδομένα για μέρα, ωστόσο τα timestamps δεν είναι απαραίτητα ισόποσα για κάθε ημέρα, ούτε στη σωστή σειρά, και τα δεδομένα είναι όλα σε μορφή text.

Αρχικά χρησιμοποιούμε το pandas, και την custom συνάρτηση load_json για να διαβάσουμε γραμμή-γραμμή το αρχείο εισόδου και να δημιουργήσουμε ένα pandas DataFrame από αυτό, με δύο στήλες, Timestamp και Value. Έχοντας δύο DataFrames αυτού του τύπου (για θερμοκρασία και υγρασία αντίστοιχα) τα συνδυάζουμε, και αφαιρούμε τα null values (θεωρούμε πως είναι ασήμαντα λίγες για να τις κάνουμε preprocess με διαφορετικό τρόπο), αποκτώντας ένα τελικό DataFrame με στήλες Timestamp, Date (μόνο ημερομηνία), Time (μόνο ώρα), Value_temp και Value_hum, όλα σε μορφή text. Τέλος μετατρέπουμε αυτο το DataFrame σε pyspark DataFrame, και κάνουμε cast τις στήλες για τις τιμές της θερμοκρασίας και υγρασίας σε double. Επιλέξαμε να κρατήσουμε την ημερομηνία σε μορφή text, καθώς είναι επαρκής για τα ζητούμενα μας.

Επεξήγηση Προσεγγίσεων / Υλοποίησης (Average Runtime: 8.9 sec)

1.1) Σε πόσες μέρες η θερμοκρασία κυμάνθηκε από 18 °C έως 22 °C;

Θεωρούμε πως το ζητούμενο είναι ο αριθμός των ημερών στις οποίες η θερμοκρασία δεν ήταν ποτέ χαμηλότερη απο 18 και ποτέ υψηλότερη απο 22 βαθμούς Κελσίου. Συνεπως η προσέγγιση μας περιλαμβάνει τη δημιουργία του DataFrame days_to_delete, το οποίο περιλαμβάνει όλα τα instances όπου η θερμοκρασία ήταν εκτός των δύο αυτών ορίων. Από αυτά διατηρούμε μόνο την στήλη της ημερομηνίας, και διαγράφουμε τα διπλότυπα, κρατώντας έτσι μόνο τις ημέρες που είχαν σε κάποιο σημείο τους τη θερμοκρασία να βρίσκεται κάτω από 18 ή πάνω από 22 βαθμούς, και στη συνέχεια τις αφαιρούμε από το σύνολο των ημερών, και μετράμε το πλήθος του συνόλου που απομένει. Το αποτέλεσμα είναι πως καμία μέρα από όσες παρέχονται στα δεδομένα δεν πληροί τις προϋποθέσεις. Αυτό είναι λογικό, καθώς οι ημέρες των δεδομένων αντιπροσωπεύουν μια κατα κανόνα χειμερινή περίοδο (με τη θερμοκρασία συχνά να βρίσκεται στους μονοψήφιους η ακόμα και να είναι αρνητική), και το εύρος θερμοκρασίας είναι πολύ μικρό (μόλις 4 βαθμοί Κελσίου) που σπανίως εμφανίζεται. Μειώνοντας τις τιμές και μεγαλώνοντας το εύρος, είναι εύκολο να παρατηρήσουμε πως ο κώδικας λειτουργεί σωστά.

1.2) Ποιες ήταν οι 10 πιο κρύες μέρες και οι 10 πιο ζεστές μέρες;

Η ερώτηση είναι ξεκάθαρη. Χρησιμοποιούμε τη συνάρτηση groupby() και avg() για να υπολογίσουμε τον μέσο όρο της θερμοκρασίας με βάση την ημέρα, και τη συνάρτηση sort() για να ταξινομήσουμε τα αποτελέσματα σε αύξουσα και φθίνουσα σειρά αντίστοιχα.

1.3) Ποιος μήνας είχε την υψηλότερη τυπική απόκλιση στις τιμές της υγρασίας;

Η ερώτηση είναι ξεκάθαρη. Η διαδικασία είναι όμοια με παραπάνω, με τη χρήση της συνάρτησης stddev() του pyspark. Για να κάνουμε group by με βάση το μήνα, χρησιμοποιούμε το substring της στήλης Date που περιλαμβάνει μόνο το έτος και τον μήνα (YYYY-MM).

1.4) Ποια ήταν η μέγιστη και ποια η ελάχιστη τιμή του δείκτη δυσφορίας;

Η ερώτηση είναι ξεκάθαρη. Δημιουργούμε ένα νέο dataframe, ίδιο με το αρχικό, με την προσθήκη της στήλης DI, στην οποία θέτουμε την τιμή του δείκτη δυσφορίας με βάση τον τύπο της εκφώνησης.

Screenshot(s):

```
Session Started.
Loaded temperature and humidity data on Spark.

Task 1.1.1: The number of days on which the temperature ranged from 18 to 22 degrees Celsius was 0 days.
Task 1.2.1: The 10 days with the lowest average temperature were the following:
2014-03-27 (2.826 C)
2014-02-14 (3.391 C)
2014-03-02 (3.857 C)
2014-03-11 (3.903 C)
2014-03-01 (4.083 C)
2014-02-13 (4.100 C)
2014-03-03 (4.132 C)
2014-03-25 (4.153 C)
2014-02-17 (4.211 C)
2014-03-05 (4.211 C)
Task 1.2.2: The 10 days with the highest average temperature were the following:
2014-05-22 (20.000 C)
2014-05-21 (17.639 C)
2014-06-08 (17.639 C)
2014-05-23 (16.257 C)
2014-06-02 (16.056 C)
2014-05-24 (15.833 C)
2014-06-03 (15.831 C)
2014-06-04 (15.710 C)
2014-06-07 (15.667 C)
2014-05-25 (15.620 C)
Task 1.3.1: The month with the highest standard deviation in humidity values was 2014-04 (17.7).
Task 1.4.1: The maximum value of the dysphoria index was 22.1.
Task 1.4.2: The minimum value of the dysphoria index was -2.33.
Session Stopped.
```

Οι απαντήσεις όλων των ερωτημάτων παρέχονται στο παραπάνω screenshot.

Θέμα 2ο: Ιστορικά Δεδομένα Μετοχών

Αρχεία εισόδου, φόρτωση και μορφοποίηση

Τα αρχεία εισόδου (agn.us.txt, ainv.us.txt, ale.us.txt) είναι σε μορφή CSV, επομένως η φόρτωση τους ως pyspark DataFrames μπορει να γίνει πολύ εύκολα μέσω της συνάρτησης spark.read().csv(). Προκύπτουν 3 DataFrames, ένα για κάθε μετοχή, με στήλες Date, Open, High, Low, Close, Volume και OpenInt, όλες σε μορφή text. Κάνουμε cast τύπους double σε όσες στήλες (Open, High, Low, Close) αναφέρονται σε χρηματικές τιμές, και integer σε όσες αντιπροσωπεύουν ακεραίους (Volume, OpenInt)

Επεξήγηση Προσεγγίσεων / Υλοποίησης (Average Runtime: 9.9 sec)

2.1) Ποιος ήταν ο μέσος όρος για τις τιμές του ανοίγματος, κλεισίματος και όγκου συναλλαγών για κάθε ημερολογιακό μήνα για κάθε μετοχή;

Η ερώτηση είναι ξεκάθαρη. Κάνουμε group by ανά μήνα (substring της στήλης Date) και υπολογίζουμε μέσες τιμές για της στήλες Open, Close και Volume. Κάνουμε την ίδια διαδικασία για κάθε ένα από τα 3 DataFrames. Κάνουμε ταξινόμηση με βάση την ημερομηνία και κάνουμε collect το αποτέλεσμα σε μορφή γραμμών.

2.2) Για πόσες ημέρες ήταν η τιμή του ανοίγματος της κάθε μετοχής πάνω από 35 δολάρια;

Η ερώτηση είναι ξεκάθαρη. Κάνουμε filter κρατώντας μόνο τις ημέρες με τιμή στη στήλη Open μεγαλύτερη από 35. Παίρνουμε τον αριθμό των ημερών που απομένουν μέσω της count(). Κάνουμε την ίδια διαδικασία για κάθε ένα από τα 3 DataFrames.

2.3) Ποιες ήταν οι μέρες με την υψηλότερη τιμή στο άνοιγμα και ποιες με την υψηλότερη τιμή στον όγκο συναλλαγών για κάθε μετοχή;

Η ερώτηση είναι ξεκάθαρη. Χρησιμοποιούμε τη συνάρτηση max_by() για να επιστρέψουμε την τιμή της στήλης Date (την ημέρα) με βάση την τιμή της στήλης Open και Volume αντίστοιχα στην ίδια γραμμή. Κάνουμε την ίδια διαδικασία για κάθε ένα από τα 3 DataFrames.

2.4) Ποιες ήταν οι χρονιές που κάθε μετοχή σημείωσε την υψηλότερη τιμή στο άνοιγμα και ποιες και τη χαμηλότερη τιμή στο κλείσιμο;

Η ερώτηση είναι ξεκάθαρη. Όπως παραπάνω, χρησιμοποιούμε την συνάρτηση max_by() και αντίστοιχα τη συνάρτηση min_by() και το substring της στήλης Date για να υπολογίσουμε το έτος. Κάνουμε την ίδια διαδικασία για κάθε ένα από τα 3 DataFrames.

Screenshot(s):

```
Session Started.
Loaded share value data for AGN, AINV and ALE on Spark.
Task 2.1.1: The average Opening Price, Closing Price and Trade Volume of AGN for each calendar month was
Month: 2005-01 | Opening Price: 30.35 USD | Closing Price: 30.23 USD | Volume: 895,649
Month: 2005-02 | Opening Price: 29.78 USD | Closing Price: 29.77 USD | Volume: 1,457,348
Month: 2005-03 | Opening Price: 31.22 USD | Closing Price: 31.21 USD | Volume: 914,340
Month: 2005-04 | Opening Price: 30.48 USD | Closing Price: 30.43 USD | Volume: 825,136
Month: 2005-05 | Opening Price: 29.64 USD | Closing Price: 29.67 USD | Volume: 1,181,446
Month: 2005-06 | Opening Price: 30.37 USD | Closing Price: 30.34 USD | Volume: 576,798
Month: 2005-07 | Opening Price: 29.65 USD | Closing Price: 29.75 USD | Volume: 861,111
Month: 2005-08 | Opening Price: 34.28 USD | Closing Price: 34.29 USD | Volume: 1,071,394
Month: 2005-09 | Opening Price: 34.64 USD | Closing Price: 34.72 USD | Volume: 951,379
Month: 2005-10 | Opening Price: 35.05 USD | Closing Price: 34.98 USD | Volume: 933,635
Month: 2005-11 | Opening Price: 32.99 USD | Closing Price: 32.99 USD | Volume: 902,174
```

```
Task 2.1.2: The average Opening Price, Closing Price and Trade Volume of AINV for each calendar month was
Month: 2005-02 | Opening Price: 8.00 USD | Closing Price: 7.96 USD | Volume: 691,247
Month: 2005-03 | Opening Price: 8.16 USD | Closing Price: 8.16 USD | Volume: 667,090
Month: 2005-04 | Opening Price: 7.96 USD | Closing Price: 7.95 USD | Volume: 815,881
Month: 2005-05 | Opening Price: 7.73 USD | Closing Price: 7.76 USD | Volume: 365,223
Month: 2005-06 | Opening Price: 8.44 USD | Closing Price: 8.49 USD | Volume: 662,943
Month: 2005-07 | Opening Price: 8.72 USD | Closing Price: 8.76 USD | Volume: 661,470
Month: 2005-08 | Opening Price: 9.12 USD | Closing Price: 9.17 USD | Volume: 873,391
Month: 2005-09 | Opening Price: 9.43 USD | Closing Price: 9.44 USD | Volume: 563,145
Month: 2005-10 | Opening Price: 8.96 USD | Closing Price: 8.94 USD | Volume: 677,502
Month: 2005-11 | Opening Price: 9.13 USD | Closing Price: 9.12 USD | Volume: 481,479
Month: 2005-12 | Opening Price: 8.91 USD | Closing Price: 8.87 USD | Volume: 685,912
Month: 2006-01 | Opening Price: 8.75 USD | Closing Price: 8.74 USD | Volume: 533,957
Month: 2006-02 | Opening Price: 9.11 USD | Closing Price: 9.10 USD | Volume: 761,886
```

```
Task 2.1.3: The average Opening Price, Closing Price and Trade Volume of ALE for each calendar month was
Month: 2005-02 | Opening Price: 30.89 USD | Closing Price: 30.89 USD | Volume: 197,887
Month: 2005-03 | Opening Price: 32.55 USD | Closing Price: 32.62 USD | Volume: 282,829
Month: 2005-04 | Opening Price: 32.23 USD | Closing Price: 32.23 USD | Volume: 201,695
Month: 2005-05 | Opening Price: 35.06 USD | Closing Price: 35.26 USD | Volume: 241,937
Month: 2005-06 | Opening Price: 38.18 USD | Closing Price: 38.25 USD | Volume: 244,023
Month: 2005-07 | Opening Price: 38.01 USD | Closing Price: 37.99 USD | Volume: 176,326
Month: 2005-08 | Opening Price: 35.74 USD | Closing Price: 35.66 USD | Volume: 237,297
Month: 2005-09 | Opening Price: 34.27 USD | Closing Price: 34.29 USD | Volume: 215,680
Month: 2005-10 | Opening Price: 33.84 USD | Closing Price: 33.76 USD | Volume: 200,767
Month: 2005-11 | Opening Price: 34.24 USD | Closing Price: 34.22 USD | Volume: 117,670
Month: 2005-12 | Opening Price: 35.51 USD | Closing Price: 35.41 USD | Volume: 119,306
Month: 2006-01 | Opening Price: 34.99 USD | Closing Price: 34.99 USD | Volume: 99,518
Month: 2006-02 | Opening Price: 34.62 USD | Closing Price: 34.71 USD | Volume: 159,199
```

Απάντηση ερωτήματος 2.1 (Το output περιλαμβάνει όλο το εύρος δεδομένων).

```
Task 2.2.1: The Opening Price of AGN was higher than 35 USD for 2071 days.
Task 2.2.2: The Opening Price of AINV was higher than 35 USD for 0 days.
Task 2.2.3: The Opening Price of ALE was higher than 35 USD for 1667 days.
Task 2.3.1a: The day on which AGN hit its highest Opening Price was 2015-07-30.
Task 2.3.2a: The day on which AINV hit its highest Opening Price was 2007-02-20.
Task 2.3.3a: The day on which ALE hit its highest Opening Price was 2017-11-01.
Task 2.3.1b: The day on which AGN hit its highest Transaction Volume was 2016-04-05.
Task 2.3.2b: The day on which AINV hit its highest Transaction Volume was 2014-02-28.
Task 2.3.3b: The day on which ALE hit its highest Transaction Volume was 2014-02-27.
Task 2.4.1a: The year during which AGN hit its highest Opening Price was 2015.
Task 2.4.2a: The year during which AINV hit its highest Opening Price was 2007.
Task 2.4.3a: The year during which ALE hit its highest Opening Price was 2017.
Task 2.4.1b: The year during which AGN hit its lowest Closing Price was 2008.
Task 2.4.2b: The year during which AINV hit its lowest Closing Price was 2009.
Task 2.4.3b: The year during which ALE hit its lowest Closing Price was 2009.
Session Stopped.
```

Απάντηση ερωτημάτων 2.2, 2.3 και 2.4.

Θέμα 3ο: Διανυκτερεύσεις σε Χώρες της Ευρώπης

Αρχεία εισόδου, φόρτωση και μορφοποίηση

Το αρχείο εισόδου (tour_occ_ninat.xlsx) είναι σε μορφή αρχείου του Microsoft Excel, το οποίο μπορεί να φορτωθεί άμεσα σε pandas DataFrame. Η πρώτη στήλη (GEO/TIME) περιλαμβάνει τα ονόματα των ευρωπαϊκων χωρών που αφορούν τα δεδομένα, ενώ οι υπόλοιπες στήλες (2006-2015) περιέχουν τον αριθμό τουριστικών διανυκτερεύσεων το αντίστοιχο έτος για κάθε χώρα. Παρατηρούμε πως ορισμένες χώρες δεν έχουν δεδομένα για κάθε έτος, και το κελί σε αυτή την περίπτωση έχει τον χαρακτήρα ":", το οποίο δυσκολεύει ιδιαίτερα τη διεκπεραίωση των ερωτημάτων. Για αυτό το λόγο κάνουμε cast όλα τα κελιά που αντιπροσωπεύουν διανυκτερεύσεις σε integer, και αντικαθιστούμε τις κενές τιμές με None/NULL. Τέλος μετατρέπουμε το pandas σε pyspark DataFrame.

Επεξήγηση Προσεγγίσεων / Υλοποίησης (Average Runtime: 50.9 sec)

3.1) Ποιος ήταν ο μέσος όρος διανυκτερεύσεων για κάθε χώρα για το χρονικό διάστημα 2007-2014;

Η ερώτηση είναι ξεκάθαρη. Για να υπολογίσουμε τον μέσο όρο διανυκτερεύσεων για κάθε γραμμή (άρα και χώρα) χρησιμοποιούμε τη συνάρτηση foreach(), η οποία εφαρμόζει μια συνάρτηση σε κάθε γραμμή του pyspark DataFrame. Η custom συναρτηση acc_mean() λαμβάνει ως είσοδο μια γραμμη DataFrame και ενα tuple που περιέχει τα έτη που μας αφορούν, σε αυτή την περίπτωση 2007-2014. Εσωτερικά, η συναρτηση κρατά ξεχωριστά το όνομα της χώρας, και δημιουργεί μια λίστα με τις τιμές (και None, όπου υπάρχουν) των διανυκτερεύσεων. Στη συνέχεια, τα None αφαιρούνται, αφήνοντας μόνο τις τιμές των ετών για τα οποία υπάρχουν δεδομένα. Αν δεν υπάρχουν δεδομένα για κανένα από τα έτη που ζητήθηκαν, η συνάρτηση εκτυπώνει την αντίστοιχη πληροφορία. Εναλλακτικά εκτυπώνεται ο μέσος όρος των διανυκτερεύσεων για τα έτη που ζητήθηκαν, και αν υπάρχουν κενές τιμές, επιπλέον εκτυπώνεται ο αριθμός των ετών για τα οποία δεν υπάρχουν δεδομένα.

3.2) Για πόσες και ποιες χρονιές ήταν ο αριθμός διανυκτερεύσεων της χώρας Ελλάδα υψηλότερος από 5 άλλες ευρωπαϊκές χώρες (της επιλογής σας);

Η ερώτηση είναι ξεκάθαρη. Οι χώρες που διαλέξαμε είναι η Γερμανία, η Βουλγαρία, το Βέλγιο, η Κροατία και η Ολλανδία. Δημιουργούμε μια λίστα με τις χώρες και ένα tuple με τα έτη που μας αφορούν, σε αυτή την περίπτωση 2006-2015. Για κάθε ένα από τα έτη, φιλτράρουμε τα δεδομένα ώστε να υπάρχουν μόνο οι χώρες που επιλέξαμε, και ταξινομούμε με βάση τον αριθμό διανυκτερεύσεων για αυτό το έτος. Αν η Ελλάδα βρίσκεται στην κορυφή, προσθέτουμε το έτος αυτό στη λίστα εξόδου, διαφορετικά συνεχίζουμε. Τέλος εκτυπώνουμε το μήκος της λίστας εξόδου (τον αριθμό των ετών) και τα έτη στα οποία η Ελλάδα είχε τις περισσότερες διανυκτερεύσεις από τις υπόλοιπες χώρες.

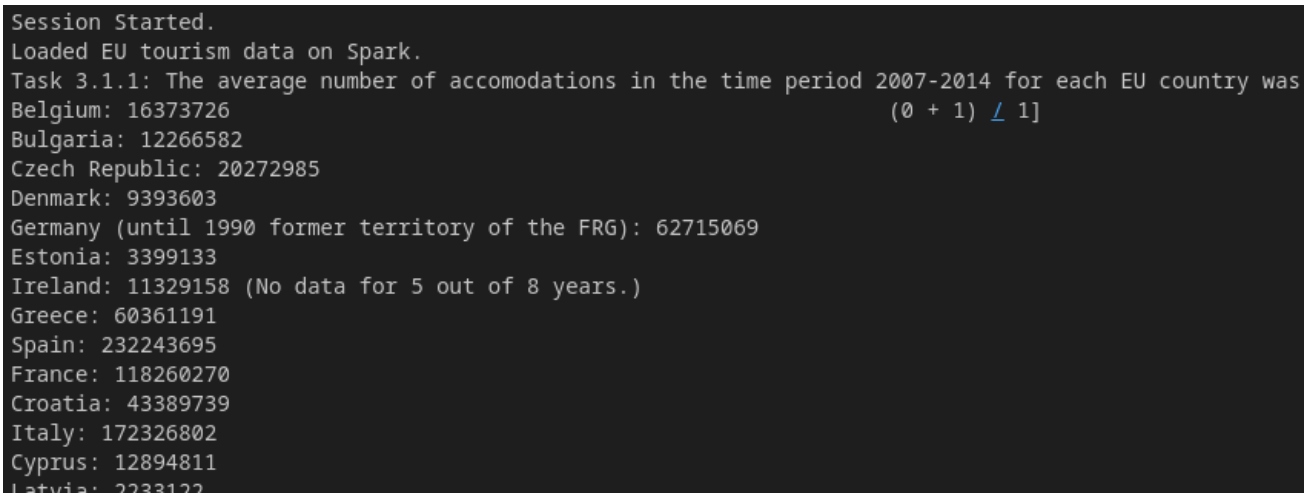
3.3) Ποιες ήταν οι χώρες με το μεγαλύτερο αριθμό διανυκτερεύσεων ανά έτος;

Θεωρούμε πως το ζητούμενο είναι να επιστρέψουμε την χώρα με τις περισσότερες διανυκτερεύσεις για κάθε έτος ξεχωριστά. Για κάθε έτος, κάνουμε sort με βάση την αντίστοιχη στήλη, και σημειώνουμε τις 3 χώρες με τις περισσότερες διανυκτερεύσεις. Παρατηρούμε πως για κάθε έτος είναι ίδιες, η Ισπανία, η Ιταλία και η Γαλλία, με αυτή τη σείρα, το οποίο αποδεικνύεται εύκολα από τα δεδομένα και είναι λογικό.

3.4) Ποια ήταν η χρονιά που η κάθε χώρα είχε το μικρότερο αριθμό διανυκτερεύσεων σε σχέση με όλες τις υπόλοιπες ευρωπαϊκές χώρες.

Θεωρούμε πως το ζητούμενο είναι να επιστρέψουμε το έτος στο οποίο κάθε χώρα είχε τις λιγότερες διανυκτερεύσεις συγκριτικά με τα υπόλοιπα έτη για την ίδια χώρα. Χρησιμοποιουμε τη συναρτηση least(), η οποία επιστρέφει τη χαμηλότερη τιμή από όλες τις στήλες για μια γραμμή. Ωστόσο, δε μας ζητείται η τιμή, αλλά το έτος, το οποίο είναι το όνομα της στήλης. Γνωρίζουμε πως δεν είναι ιδανική μέθοδος, και οφείλεται για τον σημαντικά μεγαλύτερο χρόνο εκτέλεσης αυτού του σετ ερωτημάτων, αλλά χρησιμοποιούμε ένα εμφωλευμένο for-loop για να κάνουμε αναζήτηση σε όλο το DataFrame γραμμή-γραμμή μέχρις ότου να εντοπίσουμε σε ποιά στήλη βρίσκεται η τιμή που επέστρεψε η least(). Η λύση μας είναι απαγορευτική για μεγάλων διαστάσεων DataFrames, ωστόσο δε μπορέσαμε να βρούμε κάποια καλύτερη μέθοδο για το συγκεκριμένο ερώτημα. Τέλος εκτυπώνουμε τα αποτελέσματα, κάθε χώρα και το έτος στο οποίο είχε τις λιγότερες διανυκτερεύσεις.

Screenshot(s):



Απάντηση ερωτήματος 3.1 (Χώρες χωρίς ελλιπή δεδομένα).

```
Romania: 3246491
Slovenia: 5228881
Slovakia: 4234543
Finland: 5450927
Sweden: 11371281
United Kingdom: 87177340 (No data for 2 out of 8 years.)
Iceland: 2609967
Liechtenstein: 149480
Norway: 7966946
Switzerland: No data for the time period.
Montenegro: 5443210 (No data for 6 out of 8 years.)
Former Yugoslav Republic of Macedonia, the: 714667 (No data for 1 out of 8 years.)
Serbia: 1906838 (No data for 5 out of 8 years.)
Turkey: 93587668 (No data for 6 out of 8 years.)
```

Απάντηση ερωτήματος 3.1 (Χώρες με ελλιπή δεδομένα).

```
Task 3.2.1: Greece had more accomodations than the following countries for 2 years which were: 2009, 2011
Selected countries:
Greece, Germany (until 1990 former territory of the FRG), Bulgaria, Belgium, Croatia, Netherlands
Task 3.3.1: The three EU countries with the highest accomodations for each year were:
2006: 1) Spain 2) Italy 3) France
2007: 1) Spain 2) Italy 3) France
2008: 1) Spain 2) Italy 3) France
2009: 1) Spain 2) Italy 3) France
2010: 1) Spain 2) Italy 3) France
2011: 1) Spain 2) Italy 3) France
2012: 1) Spain 2) Italy 3) France
2013: 1) Spain 2) Italy 3) France
2014: 1) Spain 2) Italy 3) France
2015: 1) Spain 2) Italy 3) France
```

Απάντηση ερωτημάτων 3.2 και 3.3.

```
Task 3.4.1: The year during which each EU country had the least accomodations were the following:
Belgium: 2009
Bulgaria: 2009
Czech Republic: 2009
Denmark: 2009
Germany (until 1990 former territory of the FRG): 2006
Estonia: 2009
Ireland: 2013
Greece: 2006
Spain: 2009
France: 2009
Croatia: 2006
Italy: 2006
Cyprus: 2009
```

Απάντηση ερωτήματος 3.4 (Το output περιλαμβάνει όλο το εύρος δεδομένων).