BU DS 561 Homework 2 - David Euijoon Kim - U66545284 - Storage Buckets

GITHUB REPOSITORY LINK: https://github.com/dk-davidekim/Google-Cloud-Computing.git

- I have included inline comments on the .ipynb file for an explanation of specific lines of code

1. The following algorithm generates a directory of 10K files with a max link count of 250. The random seed is set as 0.

    a.

```python
#!env python3
import os
import argparse
import random

def add_text(f):
    text = "Lorem ipsum dolor sit amet, \
consectetur adipiscing elit, sed do \
eiusmod tempor incididunt ut labore \
et dolore magna aliqua. Ut enim ad\n\
minim veniam, quis nostrud exercitation \
ullamco laboris nisi ut aliquip ex ea \
commodo consequat. Duis aute irure dolor \
in reprehenderit in voluptate velit esse\n\
cillum dolore eu fugiat nulla pariatur. \
Excepteur sint occaecat cupidatat non \
proident, sunt in culpa qui officia \
deserunt mollit anim id est laborum.\n<p>\n"
    f.write(text)
```

```python
def add_headers(f):
  text = "<!DOCTYPE html>\n\
<html>\n\
<body>\n"
  f.write(text)


def add_footers(f):
  text = "</body>\n\
</html>\n"
  f.write(text)

def add_link(f, lnk):
  text = "<a HREF=\""
  f.write(text)
  text = str(lnk) + ".html\""
  f.write(text)
  text = "> This is a link </a>\n<p>\n"
  f.write(text)
```

```python
def generate_file(idx, max_refs, num_files):
  directory = 'hw2_output'
  if not os.path.exists(directory):
      os.makedirs(directory)
  fname = os.path.join(directory, str(idx) + ".html")
  with open(fname, 'w', encoding="utf-8") as f:
    # how many references in this file
    add_headers(f)
    num_refs = random.randrange(0,max_refs)
    for i in range(0,num_refs):
      add_text(f)
      lnk = random.randrange(0,num_files)
      add_link(f, lnk)
    add_footers(f)
    f.close()

def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('-n', '--num_files', help="Specify the number of files to generate", type=int, default=10000)
  parser.add_argument('-m', '--max_refs', type=int, help="Specify the maximum number of references per file", default=250)
  args = parser.parse_args("")
  random.seed(0) # Add random seed = 0

  print(args.num_files, args.max_refs)
  for i in range(0,args.num_files):
    generate_file(i, args.max_refs, args.num_files)

if __name__ == "__main__":
  main()
```

    b.

2. Google Cloud Authentication using Service Account Key (JSON file)

    a.

```python
# Provide GCLOUD with the Service Account Key for Authentication Purposes
import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "/Users/davidekim/Desktop/DataScience/BU/DS561/ds-561-c49fc1ab1464.json"
```

Python

    b. I could've completed authentication through manual login, but I chose this option.

3. Import storage from google.cloud library to initialize the storage client.

a.

```python
# Initialize the Storage Client from Google Cloud
from google.cloud import storage

storage_client = storage.Client()
```

4. Used python to create the Google Cloud bucket with the name ‘bu-ds561-dk98’ and location ‘US-EAST1’

a.

```python
# Create the Bucket - Do not run if bucket already exists.
# bucket = storage_client.bucket('bu-ds561-dk98')
# bucket.location = "US-EAST1"
# bucket.create()
```

5. Open the bucket and assign it to variable ‘bucket’

a.

```python
# Open Bucket Named 'bu-ds561-dk98'
bucket = storage_client.bucket('bu-ds561-dk98')
```

6. Copy the local files to Google Cloud bucket as blobs (into a directory called ‘hw2_output’)

a.

```python
# Copy the Local Files to Google Cloud Bucket as Blobs
directory = 'hw2_output'

for file in os.listdir(directory):
    path = os.path.join(directory, file)
    bucket.blob(path).upload_from_filename(path)
```

7. While reading the files from Google Cloud bucket, it parses the html files and reads the links inside it. Then, it constructs an ‘outgoing’ graph and an ‘incoming’ graph.

a.

```python
# This Code Reads the Files in Google Cloud Bucket
# Then, Creates 'outgoing' and 'incoming' Graphs

from bs4 import BeautifulSoup

blobs = bucket.list_blobs()

outgoing = {}
incoming = {}

for blob in blobs:
    bs = BeautifulSoup(blob.download_as_text(), 'html.parser')
    links = [str(a['href'].split('.')[0]) for a in bs.find_all('a')]
    outgoing[blob.name.split('/')[1].split('.')[0]] = links
    for link in links:
        if link not in incoming:
            incoming[link] = []
        incoming[link].append(blob.name.split('/')[1].split('.')[0])
```

b. The outgoing graph looks like this (it’s a dictionary)

c.

```python
for key, value in outgoing.items():
    print(f'{key}: {value}', end='\n')
```

```
0: ['6311', '6890', '663', '4242', '8376', '7961', '6634', '4969', '7808', '5
1: ['6789', '1322', '24', '9741', '3150', '5478', '2622', '3922', '3655', '73
10: ['5678', '8126', '838', '338', '4473', '561', '4160', '9535', '4747', '33
100: ['2135', '3785', '9458', '974', '3601', '6226', '953', '4316', '9516', '
1000: ['2319', '9854', '124', '4510', '4706', '950', '6431', '4705', '1628',
1001: ['3822', '4173', '5841', '7899', '2886', '395', '3665', '9364', '5565',
1002: ['3922', '2630', '9431', '1447', '9609', '1755', '3662', '47', '7141',
1003: ['6102', '1656', '7619', '1828', '1540', '4889', '8889', '6542', '8037'
1004: ['8175', '887', '3222', '5452', '7980', '7242', '9648', '2786', '2025',
1005: ['2489', '4802', '7901', '4811', '9808', '9833', '3719', '5954', '4748'
1006: ['5092', '2162', '5410', '3188', '3619', '2590', '8300', '6609', '1805'
1007: ['454', '890', '3120', '2132', '3028', '7730', '2205', '3681', '9744',
1008: ['4263', '505', '5006', '8719', '9996', '8452', '8156', '5238', '6026',
1009: ['2379', '9598', '3140', '6539', '1061', '4945', '4846', '406', '3001',
101: ['4948', '7283', '9386', '5309', '8715', '8647', '1710', '1518', '9208',
```

d. The incoming graph looks like this (it’s a dictionary) e.

```
for key, value in incoming.items():
    print(f'{key}: {value}', end='\n')
```

```
6311: ['0', '1125', '1167', '1191', '1259', '1304', '1345', '1487', '1619', '1630'
6890: ['0', '1063', '111', '1115', '1198', '121', '1262', '1470', '1508', '1734',
663: ['0', '1054', '1110', '1280', '1286', '1303', '1317', '1468', '1668', '1734',
4242: ['0', '1108', '1198', '1229', '1286', '1409', '1570', '1585', '1632', '1707'
8376: ['0', '1093', '1127', '1200', '1882', '1975', '2026', '2047', '2086', '212',
7961: ['0', '1007', '109', '1487', '1504', '155', '1626', '1810', '1841', '1858',
6634: ['0', '1056', '1367', '1422', '1450', '1492', '1543', '1948', '2040', '2071'
4969: ['0', '1059', '1107', '1165', '1333', '1345', '139', '1407', '1597', '1664',
7808: ['0', '1082', '110', '112', '1130', '1180', '1199', '1237', '1619', '1635',
5866: ['0', '1030', '1232', '1253', '1288', '1339', '1515', '1540', '1806', '1833'
9558: ['0', '1122', '1134', '1146', '1228', '1471', '1486', '1609', '1638', '1671'
3578: ['0', '1007', '1066', '1105', '1243', '1259', '1280', '1359', '1430', '1500'
8268: ['0', '1148', '1236', '1237', '1264', '1334', '1365', '1525', '1623', '1766'
2281: ['0', '1005', '1047', '114', '1188', '1310', '1449', '1483', '155', '1677',
```

8. Find the number of links in values for each key within the items. Then, calculate the mean, median, max, min, and quintile using the Numpy library.

a.

```
# Calculate the Required Statistics of the Number of Links(Values) within the Keys.
# Calculation is Done Through the Utilization of Numpy Library

import numpy as np

outgoing_counts = [len(links) for links in outgoing.values()]
incoming_counts = [len(links) for links in incoming.values()]

out_avg = np.mean(outgoing_counts)
in_avg = np.mean(incoming_counts)

out_med = np.median(outgoing_counts)
in_med = np.median(incoming_counts)

out_max = np.max(outgoing_counts)
in_max = np.max(incoming_counts)

out_min = np.min(outgoing_counts)
in_min = np.min(incoming_counts)

out_quin = np.percentile(outgoing_counts, [20,40,60,80,100])
in_quin = np.percentile(incoming_counts, [20,40,60,80,100])
```

```python
print('outgoing average: ' + str(out_avg))
print('incoming average: ' + str(in_avg))
print('outgoing median: ' + str(out_med))
print('incoming median: ' + str(in_med))
print('outgoing maximum: ' + str(out_max))
print('incoming maximum: ' + str(in_max))
print('outgoing minimum: ' + str(out_min))
print('incoming minimum: ' + str(in_min))
print('outgoing quintile: ' + str(out_quin))
print('incoming quintile: ' + str(in_quin))
```

b.

c. Result

```
outgoing average: 123.6451
incoming average: 123.6451
outgoing median: 123.0
incoming median: 124.0
outgoing maximum: 249
incoming maximum: 188
outgoing minimum: 0
incoming minimum: 82
outgoing quintile: [ 49.  98. 149. 198. 249.]
incoming quintile: [114. 121. 126. 133. 188.]
```

d.

9. Implement the PageRank Algorithm. Explained in the comments.

```python
# PageRank Algorithm Implementation

def calculate_pagerank(outgoing, incoming): # The parameter brings the outgoing and incoming graphs

    old_PR = {page: 1/10000 for page in outgoing} # The initial pagerank for all page is 1/10000
    new_PR = dict.fromkeys(old_PR, 0) # Create a dictionary for the new pagerank (put 0s for pagerank number)
    diff = 1 # This will calculate the difference between the previous pagerank and current pagerank
    iteration = 0 # I am curious on how many iteration it goes through

    while(diff > 0.005): # Run the loop util difference is under 0.005

        for out_page in outgoing: # For all the pages in outgoing
            new_PR[out_page] = 0.15 + 0.85 * sum(old_PR[in_page] / len(outgoing[in_page]) # Calculate the algorithm provided
                                            for in_page in incoming.get(out_page))

        sum_PR = sum(old_PR.values()) # Sum of the old pagerank values
        diff = (sum(new_PR.values()) - sum_PR) / sum_PR # Find the difference percentage

        old_PR = new_PR.copy() # Move the new pagerank dictionary to variable old_PR
        iteration+=1 # Increment iteration by 1

    print('iterations: ', iteration, '; ', 'final difference: ', diff) # Print final iteration number and difference
    return sorted(old_PR.items(), key=lambda x:x[1], reverse=True)[:5] # Sort the final pagerank and get the first 5

pageranks = calculate_pagerank(outgoing, incoming) # Calculate pagerank with actual dictionaries

print(pageranks)
```

a.

```python
# Print the Top 5 Page Ranks

for pr in pageranks:
    print(pr)
```

```
('2526', 2.2521480856198735)
('6846', 2.0321429264772624)
('5971', 1.9814746529218077)
('5778', 1.9709050575289755)
('4961', 1.9683295597905284)
```

b.