Name: Jeremy Bui
Course: DS 561

Homework 2

Setup

- Created folder files locally
- Python3 generate-content.py to generate the 10,000 files
- Created bucket jeremybui_ps2 using Cloud Console
- Changed permission to all for the bucket with ‘gsutil iam ch allUsers:objectViewer gs://jeremybui_ps2’
- Uploaded all my files locally to the bucket with ‘gsutil -m cp -r files/* gs://jeremybui_ps2/files/’
- Run the calculate-pagerank.py by doing “python3 calculate-pagerank.py”
- If needed, you can change the bucket name and directory name in the main function

calculate-pagerank.py

```
import numpy as np
from google.cloud import storage
```

- Imported numpy to run calculations
- Imported google.cloud to download from storage bucket

```
def fetchLinksFromBucket(bucketName, folderName):
    storageClient = storage.Client()
    bucket = storageClient.bucket(bucketName)

    files = bucket.list_blobs(prefix=folderName)
    pageLinks = {}

    for file in files:
        pageName = file.name.split('/')[-1].split('.')[0]
        pageContent = file.download_as_text().splitlines()

        outgoingLinks = []
        for line in pageContent:
            if 'href' in line.lower() and '"' in line:
                targetPage = line.split('"')[1].split('.')[0]
                outgoingLinks.append(targetPage)
        pageLinks[pageName] = outgoingLinks
```

```
    return pageLinks
```

- Parameters:
    - buketName: name of Google Cloud bucket that has the HTML files we uploaded
    - folderName: the path to the folder in the bucket with the HTML files
- Create client to interact with Google Cloud and retrieve bucket object using bucketName
- Use list_blocks from bucket documentation to fetch all the files with folderNme prefix
- Use a dictionary for the page and number of outgoing links and then loop through and extract pageName by using file path splitting
- Create outgoingLinks to track all links in a page and then store it in dictionary

```
def computeLinkStatistics(pageLinks):
    # get counts of outgoing links for each file
    outgoingCounts = []
    incomingCounts = {page: 0 for page in pageLinks}

    # calculate outgoing links and populate incoming link counts
    for page, links in pageLinks.items():
        outgoingCounts.append(len(links))
        for link in links:
            if link in incomingCounts:
                incomingCounts[link] += 1

    # calculate statistics for outgoing links
    avgOutgoingLinks = np.mean(outgoingCounts) if outgoingCounts else
float('nan')
    medianOutgoingLinks = np.median(outgoingCounts) if outgoingCounts else
float('nan')
    maxOutgoingLinks = np.max(outgoingCounts) if outgoingCounts else
float('nan')
    minOutgoingLinks = np.min(outgoingCounts) if outgoingCounts else
float('nan')
    quintilesOutgoing = np.percentile(outgoingCounts, [20, 40, 60, 80]) if
outgoingCounts else [float('nan')] * 5
    # calculate statistics for incoming links
    incomingCountsList = list(incomingCounts.values())
    avgIncomingLinks = np.mean(incomingCountsList) if incomingCountsList
else float('nan')
    medianIncomingLinks = np.median(incomingCountsList) if
incomingCountsList else float('nan')
```

```
    maxIncomingLinks = np.max(incomingCountsList) if incomingCountsList
else float('nan')
    minIncomingLinks = np.min(incomingCountsList) if incomingCountsList
else float('nan')
    quintilesIncoming = np.percentile(incomingCountsList, [20, 40, 60,
80]) if incomingCountsList else [float('nan')] * 5

    # print statistics for outgoing and incoming links
    print("Outgoing Links - average: " + str(avgOutgoingLinks) + ",
median: " + str(medianOutgoingLinks) +
          ", max: " + str(maxOutgoingLinks) + ", min: " +
str(minOutgoingLinks) + ", quintiles: " + str(quintilesOutgoing))
    print("Incoming Links - average: " + str(avgIncomingLinks) + ",
median: " + str(medianIncomingLinks) +
          ", max: " + str(maxIncomingLinks) + ", min: " +
str(minIncomingLinks) + ", quintiles: " + str(quintilesIncoming))

    return (avgOutgoingLinks, medianOutgoingLinks, maxOutgoingLinks,
minOutgoingLinks, quintilesOutgoing,
            avgIncomingLinks, medianIncomingLinks, maxIncomingLinks,
minIncomingLinks, quintilesIncoming)
```

- Parameters:
    - pageLinks: dictionary with page name as key and list outgoing links as value
- Create empty list for number outgoing links and dictionary for page of incoming links
- Loop through the pageLinks to count outgoingCounts and update the incoming count for each incoming count
- Use numpy to calculate the average, median, max, min, and quintiles for incoming and outgoing

```
def calculatePageRank(pageLinks, tolerance=0.005,
randomJumpProbability=0.85):
    if len(pageLinks) == 0:
        print("no pages to compute pagerank.")
        return []

    pageRank = {}
    for page in pageLinks:
        pageRank[page] = 1 / len(pageLinks)

    outgoingCounts = {}
```

```
    for page, outgoing in pageLinks.items():
        outgoingCounts[page] = len(outgoing)
    incomingLinks = {}
    for page in pageLinks:
        incomingLinks[page] = []
    for page, outgoing in pageLinks.items():
        for targetPage in outgoing:
            incomingLinks[targetPage].append(page)

    delta = tolerance + 1
    iterationCount = 0

    # keep iterating until the change in pagerank values is small enough
    while delta > tolerance:
        iterationCount += 1
        newPageRank = {}

        for page in pageLinks:
            newPageRank[page] = (1 - randomJumpProbability) /
len(pageLinks)

            totalIncomingRank = 0
            for incomingPage in incomingLinks.get(page, []):
                if outgoingCounts[incomingPage] > 0:
                    totalIncomingRank += pageRank[incomingPage] /
outgoingCounts[incomingPage]
            newPageRank[page] += randomJumpProbability * totalIncomingRank

        # calculate the total change across all pages
        delta = 0
        for page in pageRank:
            delta += abs(newPageRank[page] - pageRank[page])
        pageRank = newPageRank

    # sort the pages by pagerank and get the top 5
    pageRankList = [(page, score) for page, score in pageRank.items()]
    pageRankList.sort(key=lambda x: x[1], reverse=True)
    top5Pages = pageRankList[:5]

    return top5Pages
```

- Parameters:
    - pageLinks: dictionary with page name as key and list outgoing links as value
    - Tolerance: how much page rank scores can change before iterations before ending loop
    - randomJumpProbabily: probability that it will jump to another page instead of using the normal link
- Create hashset to all have the same values with score of 1 / total pages
- Loop through items to fill in incomingLiinks with pages that the target points to
- Update page rank score until change of delta falls below tolerance
- Calculate new page rank with PR(A) = .15 + .85 * sum(incoming pageRank)
- Multiple the total rank by the probability of the random jump factor
- Update pagerank with new values until end of iteration and reverse sorted of the array
- Gather top 5 pages and return them

```
def main():
    bucketName = 'jeremybui_ps2'
    folderName = 'files'

    pageLinks = fetchLinksFromBucket(bucketName, folderName)

    if not pageLinks:
        print("no links were found.")
        return
    computeLinkStatistics(pageLinks)
    top5PagesByRank = calculatePageRank(pageLinks)
    if top5PagesByRank:
        print("Top 5 pages by PageRank:")
        for page, score in top5PagesByRank:
            print("page: " + page + ", pagerank: " + str(score))

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

- Set bucketName to your desired bucket, mine is ‘jeremybui_ps2’
- Set the folder name of the directory in the bucket, mine is ‘files’
- Call functions and print results

Output

Outgoing Links - average: 123.63273672632737, median: 123.0, max: 249, min: 0, quintiles: [49. 98. 149. 198.]

Incoming Links - average: 123.63273672632737, median: 124.0, max: 188, min: 0, quintiles: [114. 121. 126. 133.]
Top 5 pages by PageRank:
page: 2526, pagerank: 0.00023543433274959271
page: 6846, pagerank: 0.00021300280420709104
page: 5971, pagerank: 0.0002074294671529956
page: 5778, pagerank: 0.00020619141133683028
page: 1058, pagerank: 0.00020596804353742194

Cost