# Muhammad Uzair Khan - 368187

# Department of Computing CS-404 Big Data Analytics

**Class: BESE-11 & BSCS-11**

**Spring 2024**

**Lab Manual 06: Page Ranking**

**Date: 12-03-2024**

**Time: 10:00-12:50 & 14:00-16:50**

**Instructor:** Dr. Syed Imran Ali & Dr. Muhammad Daud Abdullah Asif

**Lab Engineer:** Engr. Masabah Bint E Islam

**Lab : 06 :** **Page Ranking**

**Aim:** This lab is structured to explore the foundational principles and practical applications of PageRank, a pivotal algorithm in the realm of web search and network analysis. The focus will be on both the theoretical aspects and the hands-on implementation of the PageRank algorithm, using Python as the primary tool for coding and computation. The lab aims to equip participants with a deep understanding of how web pages are ranked based on their importance and connectivity within the vast network of the internet. Emphasizing the significance of link structures and the recursive nature of PageRank, this lab will guide participants through the process of calculating the PageRank of nodes in a network. This knowledge is essential for tasks such as search engine optimization, network analysis, and understanding the dynamics of information flow on the internet. By the end of this lab, students will have gained practical experience in implementing the PageRank algorithm from scratch, as well as using library functions in Python to analyse and visualize network graphs.

**Objective:** The objectives of this lab are to provide you with a deep understanding and hands-on experience in the following areas:

* Understand the mathematical model underlying the PageRank algorithm, including concepts of link structures, damping factors, and iterative computation.
* Implement the PageRank algorithm from scratch using Python, emphasizing the practical application of theoretical concepts.
* Apply the PageRank algorithm to a sample web link graph and interpret the results to understand the importance of web pages within the network.

**Tools/Software:** IDE for python (i.e. Google Colab, Pycharm, vscode etc)

**Deliverables:** Submit a single file on LMS before the due date as communicated by Lab Engineer.

Note: Please ensure your own work, add screenshots from each step/ activity properly and submit in a Word / PDF Report Lab Report.

# Introduction to Page Ranking

PageRank is a way of assigning a numerical value to each web page, based on how many other pages link to it, and how important those pages are. The idea is that a page is more important if it is linked by many other pages, and especially by other important pages. It was developed by Google co-founders Larry Page and Sergey Brin, it is an algorithm that measures the importance of web pages. It assigns each page a numerical value based on its incoming links, considering both the quantity and quality of those links. PageRank plays a pivotal role in search engine algorithms, influencing the ranking of pages in search results. By prioritizing pages with higher authority, PageRank enhances the accuracy and relevance of search engine outcomes, contributing to a more effective and user-friendly web experience..The PageRank algorithm uses a recursive formula to calculate the PageRank of each page, based on the PageRank of the pages that link to it. The formula is:

**PR(A) = (1 - d) + d \* (PR(B) / L(B) + PR(C) / L(C) + ... + PR(N) / L(N))**

where PR(A) is the PageRank of page A, d is a damping factor (usually set to 0.85), L(B) is the number of outbound links from page B, and PR(B) / L(B) is the contribution of page B to the PageRank of page A. The formula is applied iteratively until the PageRank values converge to a stable state.

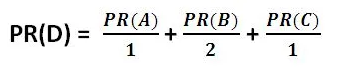
To simplify that process, let’s look at a connection of only 4 websites (A, B, C, D).

* A has 1 outgoing link (to D)
* B has 2 outgoing links (D and A)
* C has 1 outgoing link (to D)

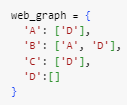
A group of circles with letters and numbers

Description automatically generated

Then the page rank for D is :



To implement the PageRank algorithm in Python, we need a way to represent the web graph, which is the network of web pages and links between them. One common way to do this is to use a dictionary, where the keys are the page names, and the values are lists of pages that link to them. For example, if we have four pages A, B, C, and D, and the links are:



To start the PageRank algorithm, we need to assign some initial values to each page. A simple way to do this is to assign a uniform value of 1 / N, where N is the number of pages in the web graph. For example, if we have four pages, we can initialize the PageRank values as:

A screenshot of a computer code

Description automatically generated

To update the PageRank values, we need to apply the formula for each page, using the current values of the other pages. We can use a loop to iterate over the web graph, and a variable to store the new values. We also need to use the damping factor, which is usually set to 0.85, to account for the possibility of random jumps from one page to another. For example, we can update the PageRank values as:

A screenshot of a computer code

Description automatically generated

To check if the PageRank values have converged to a stable state, we need to compare the old and new values, and see if they are close enough. We can use a threshold, which is a small number that indicates how much difference we can tolerate. For example, we can use a threshold of 0.001, and check for convergence as:

A screenshot of a computer code

Description automatically generated

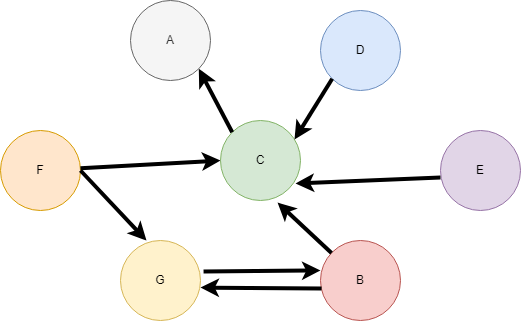
To rank the web pages, we need to sort them by their PageRank values, from highest to lowest. We can use a built-in function in Python, called sorted, which takes a list or a dictionary, and returns a sorted version of it. We can also use a lambda function, which is a short and anonymous function, to specify the sorting key, which is the PageRank value. For example, we can rank the web pages as:

A computer screen shot of a computer code

Description automatically generated

**Tasks**

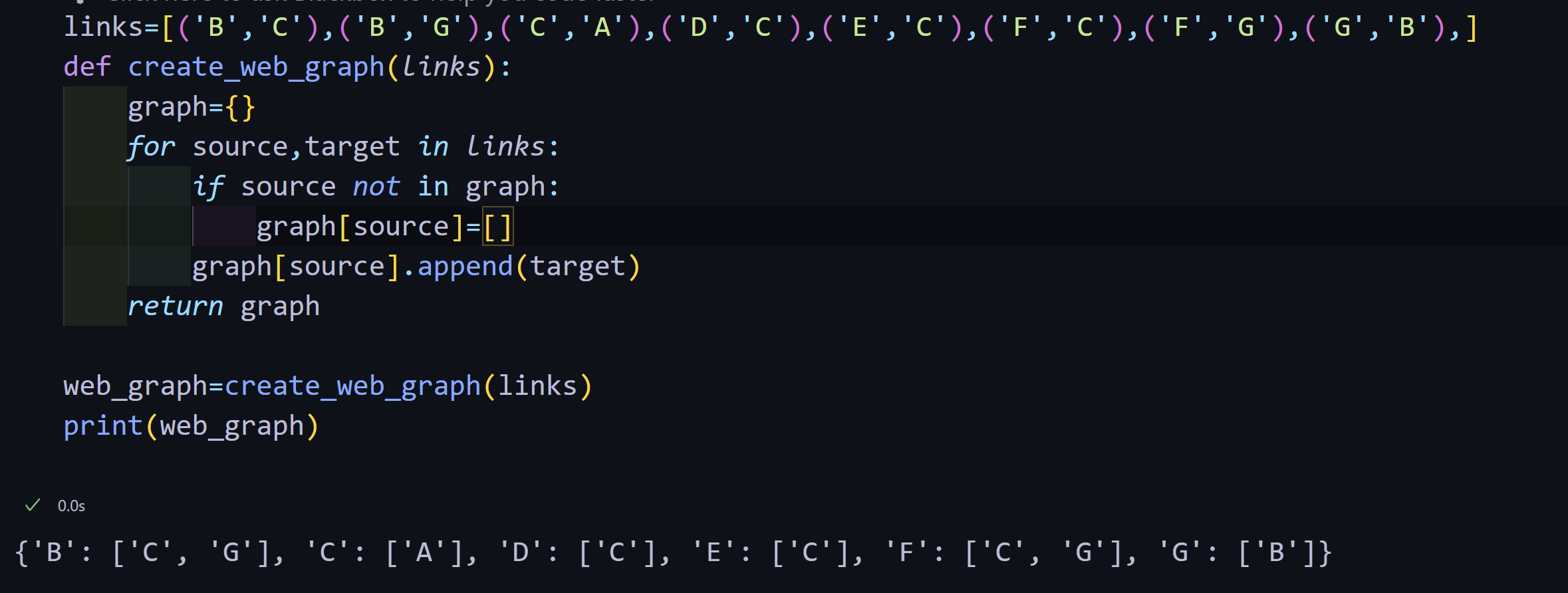
### **Implement the page rank algorithm in python for the given connections of 7 websites. Please follow all Tasks step by step to implement the algorithm.**



# Task 1

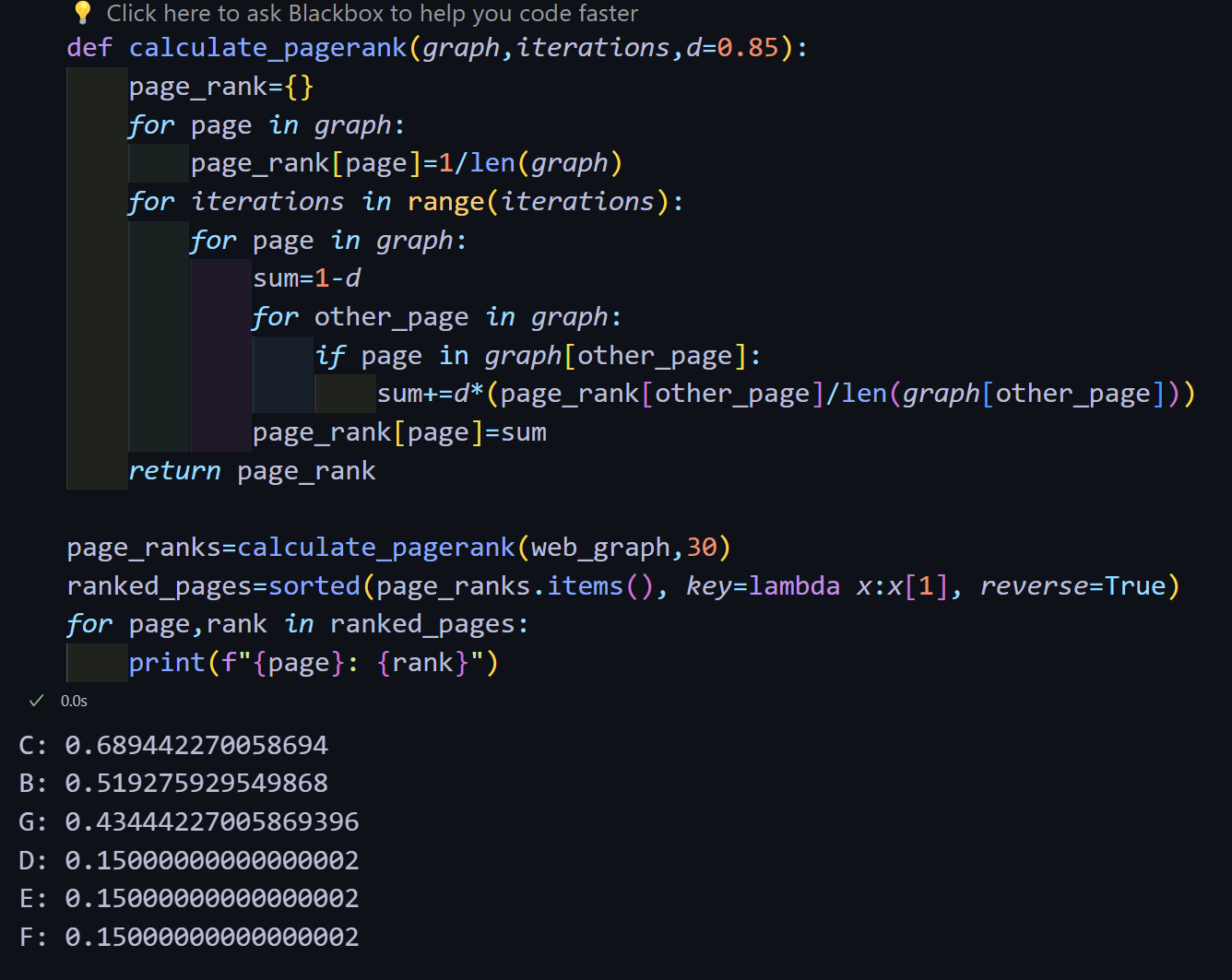
Create a function to represent the web as a graph where each node represents a webpage, and edges represent hyperlinks from one page to another. To model the web graph, you can use a dictionary in Python where each key is a unique webpage, and the corresponding value is a list of webpages that the key webpage links to. This task involves creating a function create\_web\_graph(links) where links is a list of tuples, with each tuple representing a directed link from one page to another (e.g., ('PageA', 'PageB') signifies a link from Page A to Page B). The function should return a dictionary representing the graph. Utilize the collections module for more efficient data structures if necessary.

1. Use a dictionary to represent the graph; keys are webpages, values are lists of linked webpages.
2. create\_web\_graph(links) takes a list of (source, target) tuples representing directed edges.
3. Begin with an empty dictionary to which nodes and edges will be added.
4. Add each source as a key to the dictionary, appending targets to its list, ensuring all targets are also keys.
5. Output the complete dictionary as the web graph after adding all links.



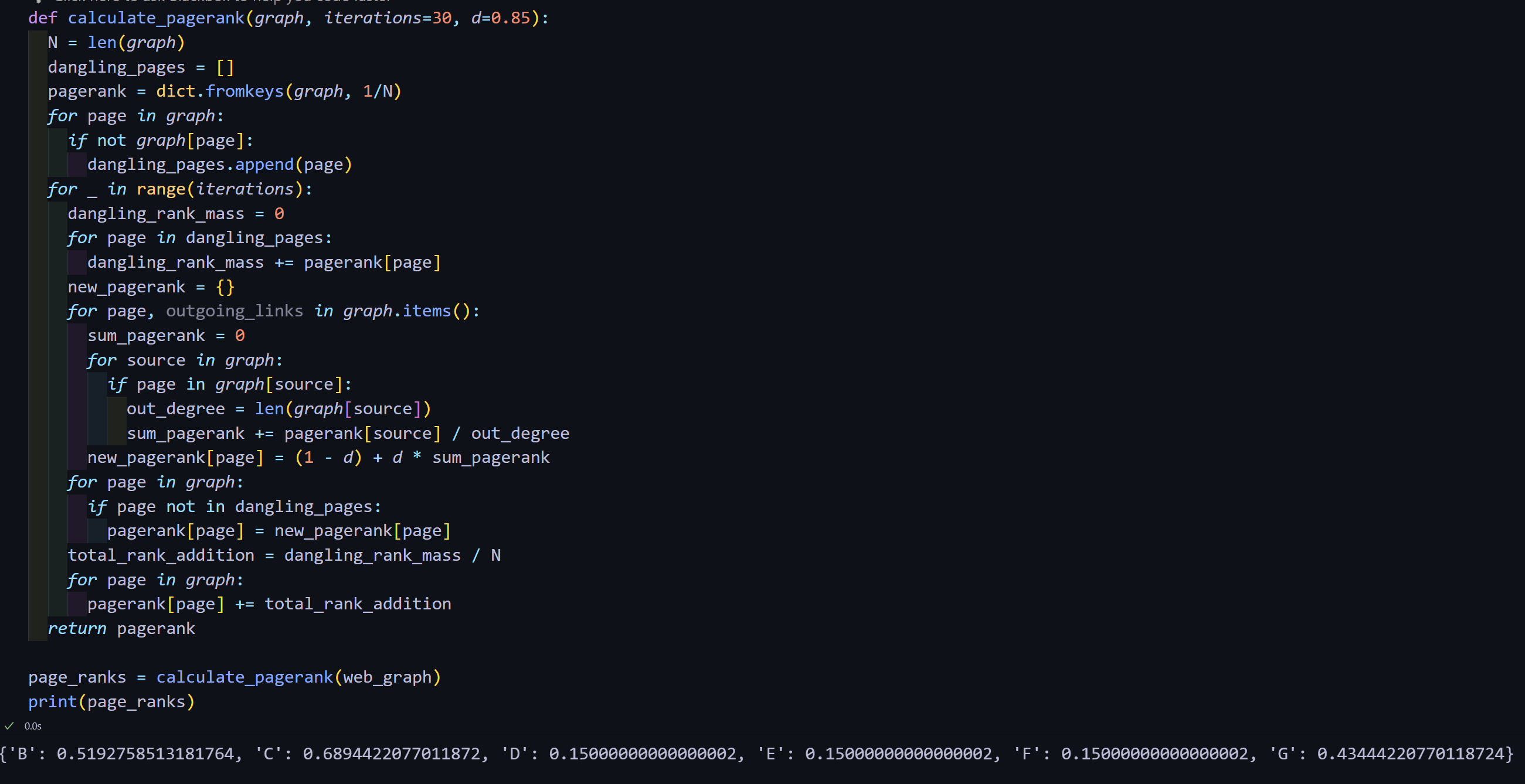
# Task 2

Implement the basic PageRank algorithm to calculate the rank of each page based on the number and quality of incoming links. This task involves creating a function calculate\_pagerank(graph, iterations, d=0.85) where graph is the web graph created in Task 1, iterations defines how many times the algorithm will run to update the PageRank values, and d is the damping factor, usually set to 0.85. Initialize the rank of each page to 1/N where N is the total number of pages. In each iteration, update the rank of each page based on the ranks of pages linking to it, considering the damping factor and the probability of jumping to a page at random. The function should return a dictionary with pages as keys and their corresponding PageRank as values.



# Task 3

Modify the PageRank calculation to handle dangling pages (pages with no outbound links) by redistributing their rank equally among all pages. Enhance the calculate\_pagerank function from Task 2 to identify dangling pages and adjust the rank calculation accordingly. After calculating the PageRank for each iteration, check for pages that do not link to any other page. Distribute the rank of these dangling pages equally to all pages in the next iteration. This ensures the total rank in the system remains constant, and the presence of dangling pages does not skew the results.



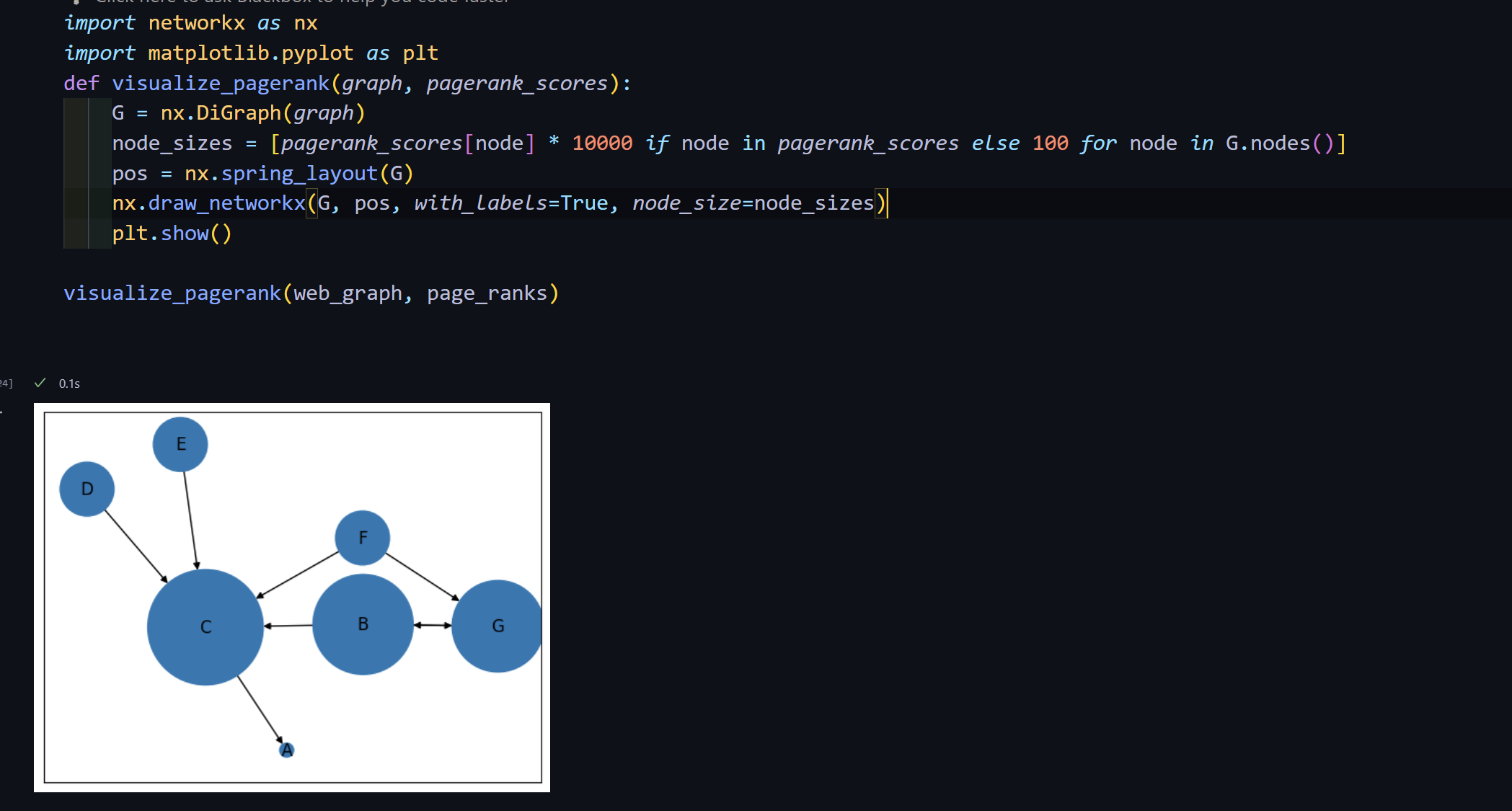
# Task 4

Implement a convergence check to stop the algorithm when the PageRank values stop changing significantly. Modify the calculate\_pagerank function to include a parameter tolerance that defines the maximum allowed change in PageRank values between iterations before considering the algorithm to have converged. After each iteration, calculate the sum of absolute differences in PageRank values between the current and previous iterations for all pages. If this sum is less than tolerance, consider the algorithm to have converged and stop the iteration. This addition will make the algorithm more efficient by preventing unnecessary calculations.



# Task 5

Create a visualization of the web graph with nodes sized according to their PageRank score to help understand the distribution of importance across pages. Use the networkx library in Python to create and visualize the graph. First, generate a NetworkX graph from your web graph dictionary. Then, use the calculate\_pagerank function to get the PageRank scores and apply these scores to determine the size of each node in the graph visualization. Utilize matplotlib for drawing the graph with networkx.draw function, where node size is proportional to its PageRank score. This task not only helps in debugging and analyzing the PageRank distribution but also provides insightful visual feedback on the structure and hierarchy of the web graph.



Reference material:

1. You can utilize the network python library for this task: <https://networkx.org/documentation/stable/reference/introduction.html#networkx-basics>
2. <https://medium.com/@arpanspeaks/custom-pagerank-implementation-in-python-and-verification-in-ms-excel-9ab6c690aaf5>