https://chat.openai.com/share/32a7f8e3-d6e1-410a-bf7e-b38305cb9427
https://github.com/Khuwaish15/AMS-325-Homework-4.git

PageRank Algorithm Implementation and Analysis

Introduction:

A vital part of search engine technology is the PageRank algorithm, which was first created by Google to rank web sites according to their significance. This paper will go over how the PageRank algorithm was implemented for a tiny web structure using a simplified model. We will also give a summary of the functionality and underlying presumptions of the code and confirm that the implementation is proper.

Implementation of PageRank Algorithm:

Function: normalize_adjacency_matrix(A)

This function creates a transition matrix (M) by normalizing the columns of an adjacency matrix (A), which is a critical step in the PageRank algorithm. The goal is to determine the likelihood that users will navigate across web pages by using links. The function generates the normalized transition matrix by using an adjacency matrix as input. The outcome is a matrix that is a key component of the PageRank calculation and shows the probability of moving from one page to another.

Function: page_rank_iteration(M, damping_factor=0.85, threshold=1e-6)

The iterative process of updating PageRank values is carried out via the PageRank Iteration function. Its input consists of the normalized transition matrix (M) plus other parameters such as the convergence threshold and damping factor. The chance of moving to any page in the web structure is represented by the damping factor, which is usually set to 0.85 in order to simulate user behavior. PageRank scores are computed once the loop is completed and PageRank values have converged. The core of the PageRank algorithm is this function.

Results and Analysis

Following the PageRank algorithm's implementation, we were able to get the following PageRank scores for a particular tiny web structure:

Page A: 0.1; Page B: 0.15; Page C: 0.1; Page D: 0.2; Page E: 0.3 F Page: 0.15
The assessed relevance of each web page inside the structure is represented by these ratings. Page E stands out in the network of pages since it has the greatest PageRank score.

Additionally, we contrasted the PageRank outcomes with the transition matrix's (M) primary eigenvector. They ought to be tightly connected, ideally. The comparison produced a

non-matching result in this particular instance. Numerous things, such as problems with convergence or errors in the PageRank iteration process, might be the cause of this.

Conclusion:

In summary, a key idea in online search and ranking is the PageRank algorithm. We confirmed the findings of a reduced version we created for a tiny web structure, however differences were found when comparing it to the primary eigenvector. The basis for additional development and analysis is provided by the functions and code structure, which are designed to be clear and easy to maintain. It could be necessary to make more adjustments and improvements in order to resolve the observed differences in PageRank values.

Game_of_life.py

A Python script called "game_of_life.py" models the dynamics of a grid of cells by using Conway's Game of Life, a classical cellular automaton. Each cell in this zero-player game has the potential to be either living or dead. Four basic laws that control cell interactions and produce complex, frequently unforeseen patterns, are followed throughout the game. Several important responsibilities, including startup, evolution, visualization, and output creation, have been included into the script. It also includes error handling to guarantee the application's resilience.

Initialization

The script starts by initializing a cell grid, setting up the scene in which the game will take place. The minimum size of the cells in this grid is 20 by 20, however you can enlarge it if necessary. At first, the states on the grid are randomly generated and might be either living (True) or dead (False). The script incorporates a method to place a predetermined pattern called the "Glider" close to the center of the grid in order to provide users a recognized starting point. A traditional pattern known as the "Glider" adds motion to the otherwise still grid by moving diagonally across it.

Evolution

The script creates a function called evolve(grid) to implement Conway's Game of Life. This function creates a new grid that represents the next evolution phase in the game using the current state of the grid as input. Four basic laws govern evolution: underpopulation, stasis, overcrowding, and reproduction. These guidelines determine whether a cell reproduces, dies, or survives to give rise to other cells. The script iterates over each cell, applying these principles and making judgments about the cell's destiny based on its eight nearby cells.

Visualization

The script's depiction of the game's progression is a key component. Throughout the simulation, Matplotlib is used to produce a visual depiction of the grid at regular intervals. To be more precise, the script makes the grid visible every tenth step so that viewers may see the dynamic changes that take place in the grid. It is simple to distinguish between living and dead cells since living cells are shown in black and dead cells are shown in white. The intriguing patterns that appear as the game progresses are captured in the resulting graphics.

Output

The screenplay also discusses the necessity of preserving the grid's ultimate condition and producing an animated depiction of the whole development. The grid's final configuration is stored as an image with the file name "game_of_life.png." At the conclusion of the simulation, this picture offers a static snapshot of the grid's condition. In addition, the script records the whole evolution process with a GIF or video, based on user desire. The resulting animation, "game_of_life.gif" or "game_of_life.mp4", shows the grid's evolution from its starting configuration to its final state at a 10 frames per second frame rate.

Error Handling

Error handling techniques are included in the script to guarantee a reliable and seamless operation. In order to ensure that the value of k, which represents the number of evolution steps, is a positive integer, it first checks the command-line input. The script will produce an understandable and informative error message before terminating the program if the input does not fulfill this condition. The script is also capable of managing errors that could occur when doing file operations, such as saving an image or creating a GIF. The proactive approach to error management improves the script's dependability and ease of usage.