Homework 6 – DFS + PageRank

Arya Rahmanian

COMP 614

December 1, 2024

CodeSkulptor Link: https://py3.codeskulptor.org/#user309_14w7HYOHg9eNCFZ.py

1. **Question 3.A.i: What is/are the base case(s) for this function? For each base case, you should clearly identify the following: The when: The condition(s) under which we fall into this base case. You can & should refer to any parameters used by name. The what: A detailed explanation of what needs to happen when we're in this base case.You can & should refer to any parameters used by name.**

When: The base case for this is the for loop condition in the "if neighbor not in parent" check. When all neighbors of the current start_node have been visited, the recursive calls for the current node to terminate.

What: No further action is taken as there are no unvisited neighbors. The function will return to the previous recursive call or terminate if it's the initial start node.

2. **Question 3.A.ii: What is/are the recursive case(s) for this function? For each recursive case, you should clearly identify the following: The when: The condition(s) under which we fall into this recursive case.You can & should refer to any parameters used by name. The what: A detailed explanation of what needs to happen when we're in this recursive case. You can & should refer to any parameters used by name. You are also welcome to assign names to local variables that you intend to define. For any recursive calls that you intend to make, you should clearly specify what the inputs will be in terms of these parameters and local variables.**

When: The recursive case is triggered when the start_node has at least one neighbor that has not been visited, mea the neighbor is not in the parent dictionary.

What: For each unvisited neighbor of start_node, the current node, or start_node, is set as the parent of the neighbor in the parent dictionary with parent[neighbor] = start_node. The recursive call to recursive_dfs is made with the same input graph, the next node to explore, and parent (the updated parent mapping).

3. **Question 3.B.i: Assume that you are not using any list comprehensions in your implementation. What is the minimum "depth" that your loop nest will need to be in order to implement this algorithm? Why? You should clearly describe the purpose of each loop.**

Minimum loop nest is 3. The first loop we get into is the while loop:

```
while delta >= epsilon:
```

Here, this loop until the pagerank values converge, meaning the total change across all nodes ranks between successive iterations is less than the threshold epsilon (set at 10^{-8}). The goal is to update the pagerank values for all nodes until they stabilize.

The second for loop:

```
for node in nodes:
```

This loop iterates over all nodes in the graph, recalculating the pagerank for each node during the current iteration of the while loop.

The last loop:

```
for neighbor in inbound_nbrs[node]
```

For a given node, this loop calculates the sum of the contributions from all its inbound neighbors to its new pagerank.

Each inbound neighbor contributes a fraction of its own pagerank, and it's weighted by the number of outbound edges (len(graph.get_neighbors(neighbor))) of the node. The sum of these contributions is used to compute the new pagerank value for the current node.

4. **Question 3.B.ii: How will you ensure that you are updating all of the page ranks "simultaneously"? In other words, how will you ensure that you are using the old $PageRank_{k-1}$ values rather than some mix of old and new values in order to compute the new $PageRank_k$ values?**

First, I will use a separate dictionary for updates. As I update, I will use a new dictionary, *new_ranks*. This will ensure that all old $PageRank_{k-1}$ values are stored in the original *ranks* dictionary, and will remain unchanged for the calculation of $PageRank_k$.

After all nodes new PageRank values have been computed and stored in *new_ranks*, the *ranks* dictionary is updated with the contents of *new_ranks* with: ranks = new_ranks

5. **Question 3.B.iii: If we think of the page ranks in terms of a random walk on the graph, what does the damping factor, d , represent? If we were to set d to 1, in what way(s) could that adversely impact the page ranks computed by the algorithm, and why? If we were to set d to 0, in what way(s) could that adversely impact the page ranks computed by the algorithm, and why?**

The damping factor, d, represents the probability that the person will continue following links. Thus, the probability that they instead jump to any random page is 1 - d.

d == 1:

d == 1 means the random walker always follows links and never teleports to a random node.

The adverse effects are if the graph has either sink nodes or disconnected components. If the graph contains sink nodes, the PageRank values will "leak" into these nodes, causing them to earn an unfairly high rank. This is because the walker gets stuck at these nodes and cannot continue. On the other hand, if the graph has disconnected components or regions with poor connectivity, some nodes may never be reached, resulting in inaccurate rankings for those nodes.

d == 0:

d == 0 means the random walker always teleports to a random node and never follows links.

The adverse effects are that all nodes will end up with the same PageRank value, 1/n, and ignores all structure of the graph. This makes the algorithm useless as we are trying to assign importance to each node/link.

6. **What does $delta_k$ represent? Why do we want to stop when $delta_k < 10^{-8}$?**

Deltak is the measure of difference between the PageRank values of all nodes at iteration k from the previous iteration, k - 1. When $delta_k < 10^{-8}$, it means the PageRank values are changing very little between iterations and the algorithm is unlikely to produce significant changes over all continuing iterations.

Discussion Question

Run PageRank on a graph built from wikipedia_articles_streamlined.txt with a damping factor of 0.85. Which ten articles have the greatest page ranks? Provide the top ten articles and their page ranks in the form of a list of tuples of (title, page rank), sorted in descending order. Do the results align with your expectations? Why or why not? Why do you think that these pages had the highest ranks?

Top 10 Articles:

Christianity: 0.0113126751178025

Soviet Union: 0.010042061855732759

European Union: 0.00955194826127956

Roman Empire: 0.00783236891615897

Europe: 0.007825005084123865

India: 0.007600195827660249

China: 0.0075346851240977

Middle Ages: 0.007236654265055148

Nazi Germany: 0.007026348927497259

Hinduism: 0.007009885604633514

The top ten pagerank values align with my expectations. These important pages are of global relevance and have a wide breadth of influence. Additionally, they are interconnected like with “Soviet Union” and “Nazi Germany”. Along with a couple major world religions and countries with international influence and economies.

Reflection

1. The two most important concepts/skills used in this assignment are the PageRank and DFS algorithm. DFS is the most, or second most, used algorithm for working with graphs, either for searching or sorting. It's a recursive algorithm that is able to reach all nodes in a graph by checking each neighbor of the current node it is looking at. As for PageRank, this algorithm is more practical for assessing the importance of nodes within a graph, an incredibly useful tool for any programmer.
2. In this example, I used these skills wikipedia article ranking, but they span way beyond that. PageRank is one of the algorithms behind search engines, as Google initially created this algorithm for their search engine. PageRank would also be insightful for social networking sites to see which pages are the most "important". DFS is more useful for navigation and topological sorting for graphs.
3. In this assignment, I believe that I did well in understanding DFS and implementing it. I have worked with graphs using DFS and BFS before, so I had a strong foundation in that. As for PageRank, it took more time for me to grasp the concept and translate the pseudocode into my final function.
4. I am comfortable with the concepts in this assignment and I would be confident teaching them to a peer. As mentioned before, I am comfortable with DFS, and while PageRank had more of a learning curve for me, it ended up being more simple than I anticipated and a really interesting algorithm that intrigued me and its use cases.