PageRank Algorithm with Markov Models and Random Sampling

1 Introduction

The PageRank algorithm is essentially a way of deciding the importance of a web page based on the number and quality of links pointing to it. It can be viewed as a Markov Chain where each state (or page) transitions to another state based on certain probabilities. The transition model of a Markov Chain provides the likelihood of moving from one state to another. In the context of PageRank, the transition model represents the probability of a user navigating from one web page to another. By repeatedly sampling states according to the transition model, we can infer the steady-state distribution (or the long-term probability of being in each state), which, in the case of PageRank, represents the importance or rank of each page. More detailed instruction about the logic of the assignment can be found at this page. To run the code, type in the terminal 'python pagerank.py corpus', where corpus is one of the three corpora of html files provided to you in the assignment folder.

2 Instructions

2.1 Function: `transition_model`

The goal of this function is to compute the probability distribution over which page to visit next, given a current page.

1. **No Outgoing Links**: If a page has no outgoing links, the random surfer has an equal probability of moving to any page. Consider how you'd distribute the probabilities across all pages in the corpus.
2. **Calculating Random Damping Probability**: The damping factor represents the probability that a surfer follows a link on a page. What's the probability that the surfer chooses a page at random instead? Use the formula:
$$\frac{1 - \text{damping factor}}{\text{total number of pages}}$$
3. **Filling Out Probability Distribution**:

```
$ python pagerank.py corpus0
PageRank Results from Sampling (n = 10000)
  1.html: 0.2223
  2.html: 0.4303
  3.html: 0.2145
  4.html: 0.1329
PageRank Results from Iteration
  1.html: 0.2202
  2.html: 0.4289
  3.html: 0.2202
  4.html: 0.1307
```

Figure 1: Example of Pagerank probabilities for corpus0.

- For each page that the current page links to: Assign the probability of following that link and assign the probability of randomly choosing that link.
- For every other page in the corpus, assign only the probability of randomly choosing that page.

2.2 Function: `sample_pagerank`

This function simulates a surfer's journey through the webpages and observes which pages they land on more frequently.

1. **Transition Model**: For each sample, determine the next page the surfer might visit by generating a transition model based on the current page.
2. **Choosing the Next Page**: Given the transition model, decide the next page the surfer visits. The Python function `random.choices` can assist in making a weighted random choice.
3. **Updating PageRank**: Keep a count of the number of times the surfer lands on each page. Use this count later to compute the probability distribution.

2.3 Function: `iterate_pagerank`

Here, you iteratively compute PageRank until the values converge.

1. **Initial PageRank**: Initially, each page should have an equal probability of being visited.
2. **Iterative Calculation**: For each page, compute its new PageRank based on the PageRanks of all other pages linking to it. Remember the formula for PageRank, which combines the random surfer model and the links pointing to the page.

3. **Checking Convergence**: After updating the PageRanks for all pages, check if their difference between old and new values is small enough. If not, repeat.

4. **Normalization**: Ensure the sum of all PageRanks is 1. If not, normalize.

3 Conclusion

Remember, the key to this exercise is understanding the transition model and how PageRank values are iteratively refined. Good luck!