Graph Mining

This task aims at exploring the PageRank algorithm on big real-world network data. The SNAP group of Stanford University has released various real-world graph datasets. Among them, you will use the Google web graph http://snap.stanford.edu/data/web-Google.html. Please refer to the first several lines in the file for the data format. Remind that we have learned how to calculate PageRank in with power iteration.

$$\mathbf{r^{(t+1)}} = \mathbf{M} \cdot \mathbf{r^{(t)}},$$

where matrix $\mathbf{M}$ is the adjacency matrix.
To address the issues caused by spider traps, we further extend the PageRank formula as:

$$\mathbf{r^{(t+1)}} = \beta\mathbf{M} \cdot \mathbf{r^{(t)}} + (1 - \beta)[1/N]_{N\times 1}, \quad (2)$$

where we assign a probability $(1 - \beta)$ to jump from the current node to all others. To address the dead-end problem, we replace the all-zero columns in the original transition matrix $\mathbf{M}$ with $[1/N]_{N\times 1}$.
The tasks are specified as follow:

1. **Implement the power iteration in matrix form as in Equation 1 without considering the dead-ends and spider traps:** Let stop criteria be $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}| < 0.02$. Calculate the rank score for all the nodes and report:

   (a) The running time and the number of iterations needed to stop.

   (b) The IDs and scores of the top-10 ranked nodes.

   Note: The matrix $\mathbf{M}$ could be too big to be stored in the memory. Please consider to use sparse matrix (e.g., use scipy.sparse in Python).

2. **Extend your PageRank code with Equation 2 to handle spider traps:** Let $\beta = 0.9$ by default and the stop criteria be $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}| < 0.02$. Run your code on the Google web data and report:

   (a) The running time and the number of iterations needed to stop.

   (b) The IDs and scores of the top-10 ranked nodes.

(c) **By varying the teleport probability** β **in** $[1, 0.9, 0.8, 0.7, 0.6, 0.5]$**, report the number of iterations needed to stop for each** β**. Explain your findings from this experiment.**

3. **Exploit dead-ends:** Before extending your codes to support dead-ends, let's first do some analysis on the current method in task 2. The stop criteria is set as $|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}| < 0.02$.

   (a) **Report the leaked PageRank score with respect to different** β **in the range** $[1, 0.9, 0.8, 0.7, 0.6, 0.5]$.

   (b) **With** $\beta = 0.9$**, report the leaked PageRank score in different iterations.**

   (c) **Explain the phenomenon you observe from the above experiments.**

4. **Implement the complete PageRank algorithm:** Extend your codes to support dead-ends. Consider to method of redistributing the leaked PageRank:

   (a) **Report the running time and the number of iterations needed to stop.**

   (b) **The IDs and scores of the top-10 ranked nodes.**