Wikipedia Web Crawler

1. Team members:
   - Michael Potts (NetId: mjpotts2, project group name: Solo)

2. I have chosen to implement a Wikipedia Web Crawler as well as the PageRank algorithm for search additional web links. In this project I will focus on creating a web crawler that can continuously search wikipedia for articles, and ensure it does not repeat article collection for previously collected articles. I plan to achieve this through the implementation of a web crawler and article retrieving program. This program will also collect a list of URLs contained in each article and decide, using the PageRank function, between a randomly selected Wikipedia page, or one of the links copied. As well, it will also create an array of linked lists to categorize alphabetically and better search through already stored documents so as to not rerun the program on pages that have been copied before. If possible, I would also like to implement a time function to record the year and month of the previous crawl through an article to allow the crawler to update pages that have since been altered. To also be careful about not causing mass webpage calls, my program with implement a sleep function of a few seconds between each page crawl. For the sake of memory preservation, I will be implementing a proof of concept model that will not download the entirety of Wikipedia, but will stop after 150 pages or so, to not overwhelm my systems storage. There are multiple examples of Web Crawler implementations in Python on the web, but ultimately, I will be making it my own, and using examples only as directional guidelines. Further, I will be implementing the Page Rank algorithm from Week 5 into my model personally.

The second part of the implementation is time dependent. If I am able, and have some time remaining, I would also like to implement a Probabilistic Topic Model on the now mined Wikipedia articles (such as PLSA or LDA). This would be quite fascinating personally to accomplish on such a large repository of data such as Wikipedia, even if completed in a basic implementation.

I would evaluate my work as personally successful if the web crawler is able to gather 150 unique pages and ensure that it does not copy a specifically not updated page twice: It should not be able to update a page for several weeks regardless as, at the time of implementation, the program would not need to be run again on specific pages as it checks updates monthly. As an additional, I would feel quite accomplished if I had time to spare, and was able to integrate at least the PLSA model from week 9, into my existing project, or if I were able to implement LDA.

3. I plan to only use python, but if speed becomes an issue, I may choose to implement the web crawler in C++. If I have additional time, I would use only python to implement PLSA or LDA.

4. For a single person, I believe the implementation of the web crawler should, or at least may, require all 20 hours of time. That being said, if the implementation is much faster, I know that the

implementation and fine tuning of a Probabilist Topic Model will require the remaining time, if not several more hours of work to properly integrate.
Implementing web crawler: 14-18 hrs
Includes:

- Article mining
- URL storage
- List of unique identifiers
- Article update function

Implementation and tuning of Page Rank: 1-2 hrs
Implementation and tuning of PLSA/LDA: – 2-8 hrs
Additional testing: 1-4 hrs
Presentation: 0.5 hrs