Project Assignment 3, COM S 535 Fall 2016
Group Members: Lei Qi & Geethanjali Jeevanatham

1. **Data Structure for Weighted Queue:**

To implement Weighted Q as descript in PA3, we used Linked List to implement it as class Weighted Queue. To make it works as expectation, for a new node, we search its correct position according to its weight at first, then insert it into the position. Because we find the proper position from end to beginning, so our weighted queue can also consider the time stamp when the node created, which means previous node will be first when its weight is same with subsequent nodes. Moreover, to improve the speed to check duplicate URL, we use a HashMap to record URLs we have in Weighted Queue up to now.

Actually, at first we tried to use Priority Queue. But after checking results carefully, Priority Queue cannot works as expected because that it cannot consider time stamp issues when two nodes' weight is same. The order in Priority Queue is not static. But if we can add a filed time stamp in node, we can also use Priority Queue. Unfortunately, it seems we cannot use additional filed to implement node as described in note.

Hence, we use Linked List to implement Weighted Queue.

2. **Pseudo code of crawler:**
    1. Input : seed node consists of seed url, weighted = 0
    2. Initialize a Weighted Queue Q and a list visited and crawlerForbiddenURL and numCrawled = 0 which record how many pages requested and max which is maximum pages to be crawled, keep a data structure HashMap<String, HashSet<String>> variable named graphAdjacencyList to make sure the number of nodes in graph is exactly max.
    3. Initialize a file named "****Graph.txt" to record graph
    4. Put seed node into Q
    5. While Q is not empty Do:
        a. Extract one node N from Q using extract method
        b. Concatenate BaseUrl with url in node N as currentPage
        c. Send a request to server at currentPage and download this page, numCrawled++, add url into visited

d. extract all links that appear in currentPage, for each link extracted
    i. compute its weight according to rule in description in PA3
    ii. created a node using link and weight
    iii. if this node's link is legal (doesn't contain #, : and not in crawlerForbiddenURL), and is not in visited, put this node into Q in proper position
    iv. graphAdjacencyList to maintain adjacency list of vertices to each vertex
e. If numCrawled % 10 == 0, sleep 5 seconds.
f. Once numCrawled = max pages are crawled then keep a temp variable saving the next highest weighted tuple to be picked and filter the adjacency list for each crawled vertex by removing every other link that has not been crawled other than the temp variable.
g. Remove the crawled nodes that has null adjacency list after the filtering step.
h. Reset numCrawled = numCrawled - the number of removed nodes in previous step.
i. crawl again till n reaches max
j. redo the filtering step till the graphAdjacencyList size equals max
k. Then do a final filtering to remove the temp variable saving the next highest weighted tuple in the graphAdjacencyList.

6. Add each record in the graphAdjacencyList to a text file which will be the max vertices graph.

Why exactly max vertices?

As above procedure described, we use a variable numCrawled (initialize it as 0) to record how many pages we have crawled (once get one page we add 1 on numCrawled) and we also keep a graphAdjacencyList to make sure the number of graph we formed is exactly max. After get one page we check if the numCrawled == max, if so, we will do filtering step and reset the

numCrawled and the crawler will continue to get pages until the graphAdjacencyList size equals max.

Hence, finally we can ensure that the graph we formed and written into file is exactly contains max vertices except case that we select a very bad seed URL or topic.

3. The output of the WikiTennisRanker is as follows:

The following results was computed on text wikiTennis.txt

For approximationFactor - 0.01
Number of Iterations of page rank algorithm to converge to epsilon – 6

top 10 pages with highest page ranks –
/wiki/United_Kingdom,
/wiki/Grand_Slam_(tennis),
/wiki/Australia,
/wiki/Romania,
/wiki/France,
/wiki/Czech_Republic,
/wiki/Serbia,
/wiki/Switzerland,
/wiki/Tennis,
/wiki/Spain

top 10 pages with highest out degree –
/wiki/Roger_Federer,
/wiki/Grand_Slam_in_tennis,
/wiki/Grand_Slam_(tennis),
/wiki/Rafael_Nadal,
/wiki/French_Open,
/wiki/Rod_Laver,
/wiki/List_of_Wimbledon_gentlemen%27s_singles_champions,
/wiki/The_Championships,_Wimbledon,
/wiki/List_of_Grand_Slam_related_tennis_records,
/wiki/Career_Golden_Slam

top 10 pages with highest in degree –
/wiki/United_Kingdom,
/wiki/Australia,
/wiki/Grand_Slam_(tennis),
/wiki/France,

Project Assignment 3, COM S 535 Fall 2016
Group Members: Lei Qi & Geethanjali Jeevanatham

/wiki/South_Africa,
/wiki/Tennis,
/wiki/The_Championships,_Wimbledon,
/wiki/Switzerland,
/wiki/Spain,
/wiki/Australian_Open

Computing Jaccard Similarity for top 100 highest page rank, out degree and in degree sets:

- Jaccard Sim of sets topInDegree and topOutDegree 0.3888888888888889
- Jaccard Sim of sets topPageRank and topOutDegree 0.30718954248366015
- Jaccard Sim of sets topPageRank and topInDegree 0.8070175438596491

For approximationFactor - 0.005
Number of Iterations of page rank algorithm to converge to epsilon – 8

top 10 pages with highest page ranks –
/wiki/United_Kingdom,
/wiki/Grand_Slam_(tennis),
/wiki/Australia,
/wiki/Romania,
/wiki/France,
/wiki/Czech_Republic,
/wiki/Serbia,
/wiki/Switzerland,
/wiki/Tennis,
/wiki/Spain

top 10 pages with highest out degree –
/wiki/Roger_Federer,
/wiki/Grand_Slam_in_tennis,
/wiki/Grand_Slam_(tennis),
/wiki/Rafael_Nadal,
/wiki/French_Open,
/wiki/Rod_Laver,
/wiki/List_of_Wimbledon_gentlemen%27s_singles_champions,
/wiki/The_Championships,_Wimbledon,
/wiki/List_of_Grand_Slam_related_tennis_records,
/wiki/Career_Golden_Slam

top 10 pages with highest in degree –
/wiki/United_Kingdom,
/wiki/Australia,
/wiki/Grand_Slam_(tennis),
/wiki/France,
/wiki/South_Africa,
/wiki/Tennis,
/wiki/The_Championships,_Wimbledon,
/wiki/Switzerland,
/wiki/Spain,
/wiki/Australian_Open

Computing Jaccard Similarity for top 100 highest page rank, out degree and in degree sets:

Jaccard Sim of sets topInDegree and topOutDegree 0.3888888888888889
Jaccard Sim of sets topPageRank and topOutDegree 0.30718954248366015
Jaccard Sim of sets topPageRank and topInDegree 0.8070175438596491

4. **The number of iterations for page rank algorithm to converge to epsilon is as below:**

The page rank was computed on wikiTennis.txt

For epsilon 0.01 the number of iterations taken by page rank to converge to 0.01 was 6 and for approximation 0.005 the number of iterations taken by page rank to converge to 0.005 was 8.

5. **MyWikiRanker**

Topic we choose: **Computer**, and topic keywords we used are **"software", "hardware", "cpu", "program", "algorithm"**. Finally we write pages into file named **WikiComputerGraph.txt**.

The following results was computed on text **WikiComputerGraph.txt**
For approximationFactor - 0.01

top 10 pages with highest page ranks –
/wiki/Computer_programming,
/wiki/Python_(programming_language),
/wiki/Functional_programming,
/wiki/Programming_paradigm,
/wiki/Programming_language,
/wiki/C_(programming_language),
/wiki/Ruby_(programming_language),
/wiki/Lisp_(programming_language),
/wiki/Java_(programming_language),
/wiki/Object-oriented_programming

Project Assignment 3, COM S 535 Fall 2016
Group Members: Lei Qi & Geethanjali Jeevanatham

For approximationFactor - 0.005

top 10 pages with highest page ranks –
/wiki/Python_(programming_language),
/wiki/Functional_programming,
/wiki/Programming_paradigm,
/wiki/Programming_language,
/wiki/C_Sharp_(programming_language),
/wiki/C_(programming_language),
/wiki/Ruby_(programming_language),
/wiki/Lisp_(programming_language),
/wiki/Java_(programming_language),
/wiki/Object-oriented_programming