COMP2521 (20T1): Assignment 2
Simple Search Engine(s)

[The specification may change, Please check the change log on this page.]

> **Change log:**
>
> - 2020-04-16 08:30 - Fixed the example on Slide 3 in *How to Get Started, Part-3*
> - 2020-04-09 15:04 - Fixed the `PageRankW` algorithm (moved the `diff` calculation outside of the inner for loop)

Objectives

- to implement simple search engines using well known algorithms like (Weighted) PageRank and rank aggregation, simplified versions for this assignment!
- to give you further practice with C and data structures (Graph ADT)

Admin

Marks	24 marks (part-1: 11 marks, part-2: 4 marks, part-3: 5 marks and 4 marks for "Style and Complexity") (14 marks towards total course mark).
Individual Assignment	This assignment is an individual assignment.
Due	08pm Friday of Week-10
Late Penalty	2 marks per day off the ceiling. Last day to submit this assignment is 05pm Monday Week-11, of course with late penalty.
Submit	Read instructions in the "Submission" section below.

Aim

In this assignment, your task is to implement simple search engines using the well known algorithm (Weighted) PageRank, simplified for this assignment, of course! You should start by reading the wikipedia entries on these topics. Later I will also discuss these topics in the lecture.

- PageRank (read up to the section "Damping factor")

The main focus of this assignment is to build a graph structure and calculate Weighted PageRanks, and rank pages based on these values. You don't need to spend time crawling, collecting and parsing weblinks for this assignment. You will be provided with a collection of "web pages" with the required information for this assignment in a easy to use format. For example, each page has two sections,

- Section-1 contains urls representing outgoing links. Urls are separated by one or more blanks, across multiple lines.
- Section-2 contains selected words extracted from the url. Words are separated by one or more spaces, spread across multiple lines.

Hint: If you use `fscanf` to read the body of a section above, you do not need to impose any restriction on line length. I suggest you should try to use this approach - use `fscanf`! However, if you want to read line by line using say `fgets`, you can assume that maximum length of a line would be 1000 characters.

Example file `url31.txt`

```
#start Section-1

url2  url34  url1 url26
url52 url21
url74  url6 url82

#end Section-1

#start Section-2

Mars has long been the subject of human interest. Early telescopic observations
revealed color changes on the surface that were attributed to seasonal vegetation
and apparent linear features were ascribed to intelligent design.
```

```
#end Section-2
```

Summary

In Part-1: You need to create a graph structure that represents a hyperlink structure of given collection of "web pages" and for each page (node in your graph) calculate Weighted PageRank and other graph properties.

In Part-2: Search Engine : Graph structure-based search engine.

In Part-3: Rank Aggregation (Hybrid search engine), you need to combine multiple ranks (for example, PageRank and tf-idf values) in order to rank pages.

Additional files: You can submit additional supporting files, `*.c`and `*.h`, for this assignment. For example, you may implement your graph adt in files `graph.c` and `graph.h` and submit these two files along with other required files as mentioned below.

Sample files

- Sample1.zip

Part-1: Graph structure-based Search Engine (11 marks)

Read the following for this part:

- Hints on "How to Implement Ass2 (Part-1)" , to be discussed in the lecture.
- Weighted PageRank Algorithm (paper)
- Hints on W^{out} : How to calculate?

Calculate Weighted PageRanks

You need to write a program in the file `pagerank.c` that reads data from a given collection of pages in the file `collection.txt` and builds a graph structure using Adjacency Matrix or List Representation. Using the algorithm described below, calculate Weighted PageRank for every url in the file `collection.txt`. In this file, urls are separated by one or more spaces or/and new line character. Add suffix `.txt` to a url to obtain file name of the corresponding "web page". For example, file `url24.txt` contains the required information for `url24`.

Example file `collection.txt`

```
url25   url31 url2
   url102   url78
url32  url98 url33
```

Simplified Weighted PageRank Algorithm you need to implement (for this assignment) is shown below. Please note that the formula to calculate PR values is slightly different to the one provided in the corresponding paper (for explanation, read Damping factor).

```
PageRankW(d, diffPR, maxIterations)

    Read "web pages" from the collection in file "collection.txt"
    and build a graph structure using Adjacency List or Matrix Representation

    N = number of urls in the collection

    For each url p_i in the collection
        PR(p_i; 0) = 1/N
    End For

    iteration = 0;
    diff = diffPR;   // to enter the following loop

    While (iteration < maxIteration AND diff >= diffPR)

        For each url p_i in the collection
```

$$PR(p_i;t+1) = \frac{1-d}{N} + d * \sum_{p_j \epsilon M(p_i)} PR(p_j;t) * W^{in}_{(p_j,p_i)} * W^{out}_{(p_j,p_i)}$$

```
    End For
```

$$diff = \sum_{i=1}^{N} Abs(PR(p_i;t+1) - PR(p_i;t))$$

```
    iteration++;
End While
```

Where,

- $M(p_i)$ is a set containing nodes (urls) with outgoing links to p_i (ignore self-loops and parallel edges)
- $W^{in}_{(p_j,p_i)}$ and $W^{out}_{(p_j,p_i)}$ are defined above (see above the link "Weighted PageRank")
- t and $(t+1)$ correspond to values of "iteration"

Note,

- For calculating $W^{out}_{(p_j,p_i)}$, if a **node k has zero out degree** (zero outlink), O_k **should be 0.5** (and not zero). This will avoid the issues related to division by zero.

Your program in `pagerank.c` will take three arguments (**d** - damping factor, **diffPR** - difference in PageRank sum, **maxIterations** - maximum iterations) and using the algorithm described in this section, calculate Weighted PageRank for every url.

For example,

```
% ./pagerank 0.85  0.00001  1000
```

Your program should output a list of urls in descending order of Weighted PageRank values (use format string `"%.7f"`) to a file named `pagerankList.txt`. We will use a tolerance of 10e-4 to check pagerank values for auto-marking. The list should also include out degrees (number of out going links) for each url, along with its Weighted PageRank value. The values in the list should be comma separated. For example, `pagerankList.txt` may contain the following:

Example file `pagerankList.txt`

```
url31, 3, 0.2623546
url21, 1, 0.1843112
url34, 6, 0.1576851
url22, 4, 0.1520093
url32, 6, 0.0925755
url23, 4, 0.0776758
url11, 3, 0.0733884
```

Sample Files for 1A

You can download the following three sample files with expected `pagerankList.txt` files. For your reference, I have also included the file "`log.txt`" which includes values of Win, Wout, etc. Please note that you do NOT need to generate such a log file.

Use format string `"%.7f"` to output pagerank values. Please note that your pagerank values might be slightly different to that provided in these samples. This might be due to the way you carry out calculations. However, make sure that your pagerank values match to at least the first 4 decimal points to the expected values. For example, say an expected value is 0.1843112, your value could be 0.1843xxx where x could be any digit.

All the sample files were generated using the following command:

```
% ./pagerank  0.85  0.00001  1000
```

- ex1
- ex2
- ex3

Part-2: Search Engine (4 marks)

Your program must use data available in two files `invertedIndex.txt` and `pagerankList.txt`, and must derive result from them for this part. We will test this program independently to your solutions for other sections.

The file named `invertedIndex.txt` contains one line per word, words are alphabetically ordered, using ascending order. Each list of urls (for a single word) are alphabetically ordered, using ascending order.

You should read the file `invertedIndex.txt` line by line, can assume max line length of 1,000 chars. Later, tokenise words (hint: use the function "`strtok`" from the `sting.h` lib) for more info see sample example-1 or sample example-2.

You can assume that you will have no duplicates for search terms, so "mars mars" is not possible.

Example file invertedIndex.txt

```
design  url2 url25 url31
mars  url101 url25 url31
vegetation  url31 url61
```

Write a simple search engine in file `searchPagerank.c` that given search terms (words) as commandline arguments, finds pages with one or more search terms and outputs (to stdout) top 30 pages in descending order of number of search terms found and then within each group, descending order of Weighted PageRank. If number of matches are less than 30, output all of them.

Example:

```
% ./searchPagerank  mars  design
url31
url25
url2
url101
```

Part-3: Hybrid/Meta Search Engine using Rank Aggregation (5 marks)

In this part, you need to combine search results (ranks) from multiple sources (say from Part-1 and assignment-1) using "**Scaled Footrule Rank Aggregation**" method, described below. All the required information for this method are provided below.

Let T1 and T2 are search results (ranks) obtained using two different criteria (say Part-1 and Part-2). Please note that we could use any suitable criteria, including manually generated rank lists.

A weighted bipartite graph for scaled footrule optimization (C,P,W) is defined as,

- C = set of nodes to be ranked (say a union of T1 and T2)
- P = set of positions available (say {1, 2, 3, 4, 5})
- **W(c,p)** is the **scaled-footrule distance** (from T_1 and T_2) of a ranking that places element 'c' at position 'p', given by

$$W(c,p) = \sum_{i=1}^{k} \left| \tau_i(c) / |\tau_i| - p/n \right|$$

where

- n is the cardinality (size) of C,
- $|T_1|$ is the cardinality (size) of T_1,
- $|T_2|$ is the cardinality (size) of T_2,
- $T_1(x_3)$ is the position of x_3 in T_1,
- k is number of rank lists.

For example,

size of T1 is 5 *size of T2 is 4*

	T1	T2
1	url1	url3
2	url3	url2
3	url5	url1
4	url4	url4
5	url2	

n is 5

C	P	W(C,P) for T1	W(C,P) for T2
url1	1	abs(1/5 - 1/5)	abs(3/4 - 1/5)
url2	3	abs(5/5 - 3/5)	abs(2/4 - 3/5)
url3	2	abs(2/5 - 2/5)	abs(1/4 - 2/5)
url4	5	abs(4/5 - 5/5)	abs(4/4 - 5/5)
url5	4	abs(3/5 - 4/5)	

$$W(c,p) = \sum_{i=1}^{k} \left| \tau_i(c)/|\tau_i| - p/n \right|$$

W(C,P) is sum of all yellow cells
(1.6 in the above example)

The **final ranking** is derived by finding **possible values of position 'P'** such that the **scaled-footrule distance is minimum**. There are many different ways to assign possible values for 'P'. In the above example P = [1, 3, 2, 5, 4]. Some other possible values are, P = [1, 2, 4, 3, 5], P = [5, 2, 1, 4, 3], P = [1, 2, 3, 4, 5], etc. For n = 5, possible alternatives are 5! For n = 10, possible alternatives would be 10! that is 3,628,800 alternatives. A very simple and obviously inefficient approach could use brute-force search and generate all possible alternatives, calculate scaled-footrule distance for each alternative, and find the alternative with minimum scaled-footrule distance.

If you use such a brute-force search, you will receive maximum of 65% for Part-3. However, you will be rewarded 100% for Part-3 if you implement a "smart" algorithm that avoids generating unnecessary alternatives, in the process of finding the minimum scaled-footrule distance. Please document your algorithm such that your tutor can easily understand your logic, and clearly outline how you plan to reduce search space, otherwise you will not be awarded mark for your "smart" algorithm! Yes, it's only 35% of part-3 marks, but if you try it, you will find it very challenging and rewarding.

Write a program `scaledFootrule.c` that aggregates ranks from files given as commandline arguments, and output aggregated rank list with minimum scaled footrule distance.

- **How to Get Started, Part-3**

Example, file rankA.txt

```
url1
url3
url5
url4
url2
```

Example, file rankD.txt

```
url3
url2
url1
url4
```

The following command will read ranks from files "rankA.txt" and "rankD.txt" and outputs minimum scaled footrule distance (using format %.6f) on the first line, followed by the corresponding aggregated rank list.

```
% ./scaledFootrule   rankA.txt  rankD.txt
```

For the above example, there are **two possible answers, with minimum distance of 1.400000**.

Two possible values of P with minnimum distance are:

```
C = [url1, url2, url3, url4, url5]
P = [1, 4, 2, 5, 3] and
P = [1, 5, 2, 4, 3]
```

By the way, you need to select any one of the possible values of P that has minium distance, so there could be multiple possible answers. Note that you need to output only one such list.

One possible answer for the above example, for `P = [1, 4, 2, 5, 3]` :

```
1.400000
url1
url3
url5
url2
url4
```

Another possible answer for the above example, `P = [1, 5, 2, 4, 3]` :

```
1.400000
url1
url3
url5
url4
url2
```

Please note that your program should also be able to handle multiple rank files, for example:

```
% ./scaledFootrule   rankA.txt  rankD.txt  newSearchRank.txt  myRank.txt
```

Submission

Additional files: You can submit additional supporting files, `*.c` and `*.h`, for this assignment.

IMPORTANT: Make sure that your additional files (*.c) DO NOT have "main" function.

For example, you may implement your graph adt in files `graph.c` and `graph.h` and submit these two files along with other required files as mentioned below. However, make sure that these files do not have "main" function.

I explain below how we will test your submission, hopefully this will answer all of your questions.

You need to submit the following files, along with your supporting files (*.c and *.h):

- pagerank.c
- searchPagerank.c
- scaledFootrule.c

Now say we want to mark your `pagerank.c` program. The auto marking program will take all your supporting files (other *.h and *.c) files, along with `pagerank.c` and execute the following command to generate executable file say called pagerank. Note that the other four files from the above list (`searchPagerank.c` and `scaledFootrule.c`) will be removed from the dir:

```
% gcc -Wall -lm -std=c11  *.c  -o pagerank
```

So we will **not use your Makefile** (if any). The above command will generate object files from your supporting files and the file to be tested (say `pagerank.c`), links these object files and generates executable file, say `pagerank` in the above example. Again, please make sure that you **DO NOT have main function in your supporting files** (other *.c files you submit).

We will use similar approach to generate other executables (from `searchPagerank.c` and `scaledFootrule.c`).

How to Submit

Go to the following page, select the tab "Make Submission", select "Browse" to select all the files you want to submit and submit ising "Submit" button. The submission system will try to compile each required file, and report the outcome (ok or error). Please see the output, and correct any error. If you do not submit a file(s) for a task(s), it will report it as an error(s).

You **can now submit this assignment**, click on "Make Submission" tab, and follow the instructions.

Plagiarism

This is an individual assignment. You are not allowed to use code developed by persons other than you. In particular, it is not permitted to exchange code or pseudocode between students. You are allowed to use code from the course material (for example, available as part of the labs, lectures and tutorials). If you use code from the course material, please **clearly acknowledge** it by including a comment(s) in your file. If you have questions about the assignment, ask your tutor.

Before submitting any work you should read and understand the sub section named ***Plagiarism*** in the section "Student Conduct" in the course outline. We regard unacknowledged copying of material, in whole or part, as an extremely serious offence. For further information, see the course outline.

-- end --