# COMP9024: Assignment 2
# Simple Graph structure-based Search Engine

[The specification may change. A notice on the class web page will be posted after each revision, so please check class notice board frequently.]

> **Change log:**
>
> - No entry as yet!

## Objectives

- to implement a simple search engine based on the well known PageRank algorithm (simplified).
- to give you further practice with C and advanced data structures (Graph and BST ADTs)

## Admin

| | |
|---|---|
| **Marks** | 15 marks towards total course marks |
| **Due** | 11pm Sunday 03 Feb 2019. |
| **Late Penalty** | 2 marks per day off the ceiling.<br>Last day to submit this assignment is 5pm Tuesday 05 Feb 2019, of course with late penalty. |
| **Submit** | Read instructions in the "Submission" section below. |

## Aim

In this assignment, your task is to implement a simple search engine using the well known algorithm PageRank, simplified for this assignment, of course!. You should start by reading the wikipedia entries on the topic. 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, calculate PageRanks and rank pages based one 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:* You can assume that maximum length of a line would be 1000 characters. You need to use a dynamic data structure(s) to handle words in a file and across files, no need to know max words beforehand.

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
```

Your tasks in summary:

- *Calculate PageRanks*: 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 PageRank value and other graph properties.
- *Inverted Index*: You need to create "inverted index" that provides a list of pages for every word in a given collection of pages.
- *Search Engine*: Your search engine will use the given inverted index to find pages where query term(s) appear and rank these pages using their PageRank values (see below for more details)

## How to get started Hints and Sample files

- Hints on "How to Implement Ass2" (HowToImplement-Ass2.pdf), will be discussed in the lecture.
- Sample files for How to Get Started (ass2-getting-started.zip), will be discussed in the lecture.
- Sample1.zip

## 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.

---

# Part-A: Calculate 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 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 PageRank Algorithm (for this assignment)

```
PageRank(d, diffPR, maxIterations)

    Read "web pages" from the collection in file "collection.txt"
    and build a graph structure using Adjacency List 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)
        iteration++;

        PR(p_i; t+1) = (1-d)/N + d * sum_{p_j ∈ M(p_i)} PR(p_j; t)/L(p_j)

            where,
              - M(p_i) is a set containing links(urls) pointing to pi
                    (ignore self-loops and parallel edges)
              - L(p_j) is out degree of p_j
              - t+1 corresponds to value of "iteration"

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

    End While
```

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 PageRank for every url.

For example,

```
% pagerank 0.85  0.00001  1000
```

Your program should output a list of urls in descending order of PageRank values (use format string `"%.7f"`) to a file named `pagerankList.txt`. The list should also include out degrees (number of out going links) for each url, along with its 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 Part-A**

You can download the following three sample files with expected `pagerankList.txt` files.

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 say first 6 decimal points to the expected values. For example, say an expected value is 0.1843112, your value could be 0.184311x where x could be any digit.

All the sample files were generated using the following command:

```
% pagerank 0.85 0.00001 1000
```

- aEx1
- aEx2
- aEx3

## Part-B: Inverted Index

You need to write a program in the file named `inverted.c` that reads data from a given collection of pages in `collection.txt` and generates an "inverted index" that provides a sorted list (set) of urls for every word in a given collection of pages. Before inserting words in your index, you need to "normalise" words by,

- removing leading and trailing spaces,
- converting all characters to lowercase,
- remove the following punctuation marks, if they appear at the end of a word: '.' (dot), ',' (comma), ';' (semicolon), ? (question mark)

In each sorted list (set), duplicate urls are not allowed. Your program should output this "inverted index" to a file named `invertedIndex.txt`. One line per word, words should be alphabetically ordered, using ascending order. Each list of urls (for a single word) should be alphabetically ordered, using ascending order.

Example file `invertedIndex.txt`

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

**Note**: for this part, in your output file, on each line, a word and urls must be separated by one (or more) spaces. The testing program will ignore additional spaces.

## Part-C: Search Engine

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 PageRank. If number of matches are less than 30, output all of them.

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

**Note:** For this part,

- each line in "invertedIndex.txt" contains - a word and the corresponding urls separated by one (or more) spaces. Your program for Part-C needs to be able to handle such an input. Please see the sample program provided "exTkns.c" .
- each line in "pagerankList.txt" contains - url, out-degree and pagerank. To simplify your task, you can assume that they are separated by ", " - that is a comma and one space.

Example:

```
% searchPagerank  mars  design
url31
url25
```

## 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
- inverted.c
- searchPagerank.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 two files from the above list (`inverted.c` and `searchPagerank.c`) will be removed from the dir:

```
% gcc -Wall -lm -Werror *.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 two executables (for `inverted.c` and `searchPagerank.c`).

## How to Submit

Go to the following submission page, select the tab named "Make Submission", and follow the instructions. 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).

- Assignment 2 Submission Page

---

# Plagiarism

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.

Your program must be entirely your own work. Plagiarism detection software compares all submissions pairwise (including submissions for similar projects in previous years, if applicable) and serious penalties will be applied, particularly in the case of repeat offences.

***Do not copy from others; do not allow anyone to see your code, not even after the deadline***

Please refer to the on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Procedure

Before submitting any work you should read and understand the sub section named ***Plagiarism*** 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 --