Counting mutual web linkage occurrences

IN3200/IN4200 Home Exam 1, Spring 2020

Note: Each student should independently program the required code and write her/his own short report. The detailed submission information can be found at the end of this document.

1 Introduction

1.1 Web graph

The linkage situation among a group of webpages can be illustrated by a **web graph**, which shows how the webpages are directly connected by *outbound links*. An example of a web graph can be found in Figure 1. In this particular example, there are eight webpages, where webpage No. 1 is directly linked to webpage Nos. 2 & 3, webpage No. 2 is directly linked to webpage No. 4, and so on.

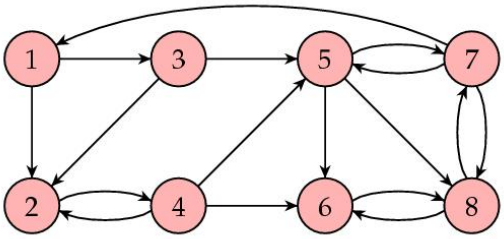

Figure 1: A very simple graphical representation example of eight webpages that are linked to each other.

1.2 Mutual web linkage

One way to characterise a webpage group is to analyse how likely two webpages are directly linked to a third common webpage. For that purpose, it is necessary to count the number of occurrences for **mutual web linkage**, that is, two webpages i and j (outbound) are both directly linked to webpage k (inbound), where $i \neq j \neq k$. Take for instance the web graph in Figure 1. There are in

total 13 occurrences of mutual web linkage, more specifically,

$$\begin{aligned}
&(w1, w3) \Rightarrow w2,\ (w1, w4) \Rightarrow w2,\ (w3, w4) \Rightarrow w2,\\
&(w3, w4) \Rightarrow w5,\ (w3, w7) \Rightarrow w5,\ (w4, w7) \Rightarrow w5,\\
&(w4, w5) \Rightarrow w6,\ (w4, w8) \Rightarrow w6,\ (w5, w8) \Rightarrow w6,\\
&\qquad\qquad\qquad\qquad\qquad\qquad (w5, w8) \Rightarrow w7,\\
&(w5, w6) \Rightarrow w8,\ (w5, w7) \Rightarrow w8,\ (w6, w7) \Rightarrow w8.
\end{aligned}$$

Another statistical quantity of interest is the number of times that a webpage is involved as outbound for the mutual web linkages. For the web graph in Figure 1, webpage 4 is involved six times, webpage 5 is involved five times, webpages 3 and 7 are each involved four times, webpage 8 is involved three times, webpages 1 and 6 are each involved twice.

2 Data storage formats for the web graph

2.1 2D table

The most convenient data storage format for a web graph is a 2D table (2D array), which is of dimension $N \times N$, where N denotes the number of webpages. The values in the table are either "0" or "1". If the value on row i and column j is "1", it indicates a direct link from webpage j (outbound) to webpage i (inbound). For instance, the corresponding 2D table for the web graph in Figure 1 is as follows:

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}$$

2.2 Compressed rows

When the number of webpages N is large, and if the total number of direct links (denoted as N_{links}) is relatively small, then most of the values in the 2D table will be zero. This can result in a huge waste of storage, due to the huge number of zero values. Instead, a **compressed row storage** (CRS) format can be adopted. More specifically, two 1D arrays of integer values are enough. The `col_idx` array is of length N_{links} and stores, row by row, the column indices corresponding to all the direct links. The `row_ptr` array is of length $N+1$, which can be used to "dissect" the `col_idx` values, row by row. For more information about CRS, please read Section 3.6.1 of the textbook. (Note: There is a typo in the textbook about the length of `row_ptr`, which should be $N + 1$ (not N).)

3 Assignments of the home exam

3.1 Reading a web graph from file

Assuming that a web graph is stored in a text file (the file format will be described below), you should implement two functions with the following syntax:

- `void read_graph_from_file1 (char *filename, int *N, char ***table2D)` which opens a text file (filename prescribed as input), allocates a 2D table of correct dimension and reads the web graph info into the table.
- `void read_graph_from_file2 (char *filename, int *N, int *N_links, int **row_ptr, int **col_idx)` which has the same purpose as above except that the CRS format is used. (Note: In this function, you should avoid internally allocating/using a 2D table.)

3.2 Counting mutual web linkages

You should implement two functions for counting the total number of mutual webpage linkage occurrences, as well as the number of involvements per webpage as outbound for such mutual linkage occurrences. (See above for the definitions.)

- `int count_mutual_links1 (int N, char **table2D, int *num_involvements)`
- `int count_mutual_links2 (int N, int N_links, int *row_ptr, int *col_idx, int *num_involvements)`

Both functions should return the total number of mutual webpage linkage occurrences. In addition, the array `num_involvements` is assumed to be already allocated (of length N) beforehand and should contain the number of involvements per webpage when the functions are finished.

3.3 OpenMP parallelisation of the two `count_mutual_links` functions

You can choose to insert OpenMP directives directly into the above two functions (if so, make sure that the two functions can be compiled with and without OpenMP enabled). Alternatively, you can make separate implementations of the OpenMP version.

3.4 Finding top webpages regarding involvements in mutual linkages

You should implement the following function

```
void top_n_webpages (int num_webpages, int *num_involvements,
int n)
```

which finds the top n webpages with respect to the number of involvements in mutual linkages, and print out these webpages and their respective numbers of involvements.

Required for IN4200 students: Implement the OpenMP parallelisation of the `top_n_webpages` function, in addition.

3.5 Test program

You should write a test program that makes use of all the functions implemented above. The time usage of each function should also be measured. A filename is supposed to be given on the command line. You can choose to make the test program compilable with and without OpenMP enabled, or alternatively make a separate OpenMP enabled test program.

4 File format of a web graph

It can be assumed that a text file containing a web graph (that is, the webpage linkage connection information) has the following format:

- The first two lines both start with the `#` symbol and contain free text (listing the name of the data file, authors etc.);
- Line 3 is of the form "`# Nodes: integer1 Edges: integer2`", where `integer1` is the total number of webpages, and `integer2` is the total number of links. (Here, nodes mean the same as webpages, and edges mean the same as links.)
- Line 4 is of the form "`# FromNodeId ToNodeId`";
- The remaining part of the file consists of a number of lines, the total number equals the number of links. Each line simply contains two integers: the index of the outbound webpage and the index of the inbound webpage;
- Some of the links can be self-links (same outbound as inbound), these should be excluded (not used in the data storage later);
- Note: the webpage indices start from 0 (C convention).
- It can be assumed that each web link is uniquely listed, that is, there will NOT be multiple text lines describing the same web link.
- You may NOT assume that the links are sorted with respect to `FromNodeId`.
- For each `FromNodeId`, you may NOT assume that the links are sorted with respect to `ToNodeId`.

An example web graph file that contains the linkage connection information shown in Figure 1 is as follows:

```
# Directed graph (each unordered pair of nodes is saved once): 8-webpages.txt
# Just an example
# Nodes: 8 Edges: 17
# FromNodeId    ToNodeId
0       1
0       2
1       3
2       4
2       1
3       4
3       5
3       1
4       6
4       7
4       5
5       7
6       0
6       4
6       7
7       5
7       6
```

An example real-world web graph file (rather large) can be downloaded from `https://snap.stanford.edu/data/web-NotreDame.html`.

5 Submission

Each student should submit a single tarball (`.tar`) or a single zip file (`.zip`). Upon unpacking/unzipping it should produce a folder named `IN3200_HE1_xxx` or `IN4200_HE1_xxx`, where `xxx` should be the candidate number of the student (can be found in StudentWeb). Inside the folder, there should be the following files:

- One `*.c` file for the implementation of each of the functions. The filename should be same as the function name.
- A `README.txt` file explaining how the compilation should be done, with additional comments if relevant. You are strongly advised to make sure that your implemented code can be compiled on a standard Linux computer.
- Preferably a Makefile.
- A PDF file containing a short report that describes the most important algorithmic and programming info, as well as the time measurements of the OpenMP parallelised functions (when the number of OpenMP threads is varied.)

6 Grading

The grade of the submission will constitute 20% of the final grade of IN3200/IN4200. Grading of the submission will be based on the correctness, conformability (of filenames and function syntax), readability and speed of the implementations,

in addition to the quality of the short report. **Hint:** In order to make sure that your code compiles on a standard Linux computer, please test the compilation on any of Ifi's Linux servers (e.g. `login.ifi.uio.no`.)