COP 3530 Programming Assignment -1: Simplified Page Rank Algorithm

In late 90's as the number of webpages on the internet were growing exponentially different search engines were trying different approaches to rank the webpages. At Stanford, two computer science PhD students, Sergey Brin and Larry Page were working on the following questions: How can we trust information? Why are some web pages more important than others? Their research led to the formation of the Google search engine. In this programming assignment, you are required to implement a simplified version of the original PageRank algorithm on which Google was built.

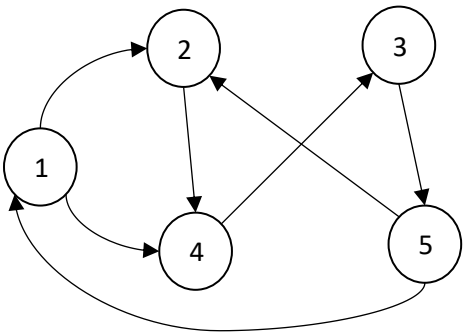
Representing the Web as a Graph

The idea that the entire internet can be represented as a graph. Each node represents a webpage and each edge represents a link between two webpages. This graph can be implemented as an Adjacency Matrix or an Adjacency List. **Feel free to use any data structure.**

Now for the sake of simplicity, we are explaining the assignment in the form of an Adjacency Matrix. We represent the graph in the form of |V|x|V| matrix where |V| is the total number of vertices in the graph. This is mapped to the webpages in the entire internet. Thus, if there is an edge from V_i to V_j (the from_page points to_page), we have the value in our adjacency matrix $M_{ij} = 1$ and 0 otherwise.

M=

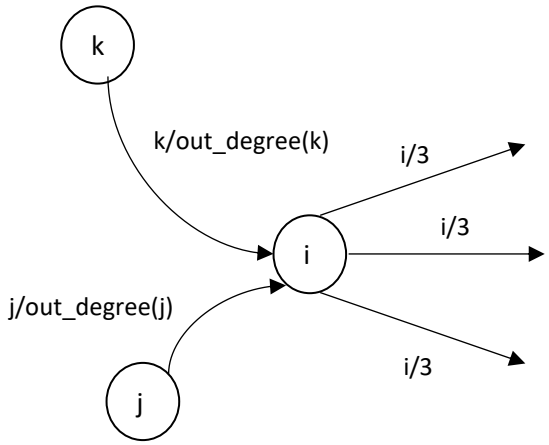
	1	2	3	4	5
1	0	1	0	1	0
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	1	0	0
5	1	1	0	0	0



Core Idea of PageRank

1. Important web pages will point to other important webpages.
2. Each page will have a score and the results of the search will be based on the page score (called page rank).

Rank(i) = j/out_degree(j) + k/out_degree(k)



Each webpage is thus a node in the directed graph and has incoming edges and outgoing edges. Each node has a rank. According to PageRank, this rank is equally split among the node's outgoing links and this rank is equal to the sum of the incoming ranks. The rank is based on the indegree (the number of nodes pointing to it) and the importance of incoming node. This is important considering let's say you create your personal website and have a million links to other pages of importance. If this was not the case and rank used out links, we can easily dupe the algorithm. Therefore, the rank is based on in-links.

Goal

In this assignment, you need to compute the rank of the webpages using a Simplified Algorithm explained in the example below.

Input

Line 1 contains the number of lines (n) that will follow and the number of power iterations you need to perform. Each line from 2 to n will contain two URL's – *from_page* *to_page* separated by a space. This means from_page points to the URL to_page.

Output

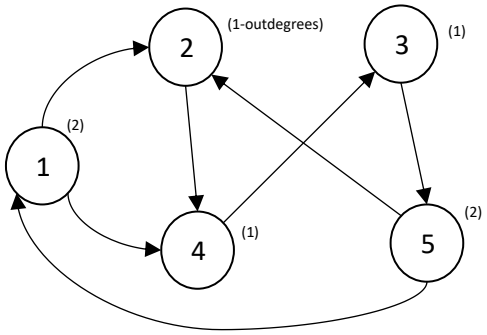
Print the PageRank of all pages after n powerIterations in ascending alphabetical order of webpage. Also, round off the rank of the page to two decimal places.

<u>Explanation through a Core Example</u>

Input:

7 2

google.com	gmail.com
google.com	maps.com
facebook.com	ufl.edu
ufl.edu	google.com
ufl.edu	gmail.com
maps.com	facebook.com
gmail.com	maps.com



Output

facebook.com 0.20
gmail.com 0.20
google.com 0.10
maps.com 0.30
ufl.edu 0.20

Step 1: Mapping for Simplicity (Optional but you will need a mechanism to store unique vertices)

Use a map/array to map the URL's with a unique id

1	google.com
2	gmail.com
3	facebook.com
4	maps.com
5	ufl.edu

	Data Structure 1
1	google.com
2	gmail.com
3	facebook.com
4	maps.com
5	ufl.edu

Step 2: Graph Representation and Page Rank

In page rank, the equation for your graph is given as follows:-

Rank of a Page, r= M.r where,

M is the matrix with values given by the following:

$M_{ij} = 1/d_j$ if there is an edge from V_j to V_i (d_j is the outdegree of node j)

0 otherwise

For our graph, the adjacency matrix, M will look like:

	1	**2**	**3**	**4**	**5**
1	0	0	0	0	$1/d_5$
2	$1/d_1$	0	0	0	$1/d_5$
3	0	0	0	$1/d_4$	0
4	$1/d_1$	$1/d_2$	0	0	0
5	0	0	$1/d_3$	0	0

Step 3: Power iteration, r(t+1)=M.r(t)

This means that a rank of the webpage at time t+1 is equal to the rank of that page at time t multiplied by matrix, M. To achieve this, we create our matrix M based on input. Next, we initialize r(t) which is a matrix of size |V|x1 and consists of the ranks of every webpage. We initialize r(t) to 1/|V|. Next we compute power_iterations based on our input.

r(0)

	1
1	1/5
2	1/5
3	1/5
4	1/5
5	1/5

M

	1	2	3	4	5
1	0	0	0	0	$1/d_5$
2	$1/d_1$	0	0	0	$1/d_5$
3	0	0	0	$1/d_4$	0
4	$1/d_1$	$1/d_2$	0	0	0
5	0	0	$1/d_3$	0	0

r(t+1)=r(0+1)=r(1) =M.r(0)=

	1	2	3	4	5			1			1
1	0	0	0	0	1/2		**1**	1/5	=	**1**	1/10
2	1/2	0	0	0	1/2		**2**	1/5		**2**	1/5
3	0	0	0	1	0	x	**3**	1/5		**3**	1/5
4	1/2	1	0	0	0		**4**	1/5		**4**	3/10
5	0	0	1	0	0		**5**	1/5		**5**	1/5
	M			**x**			**r(0)**		**=**	**r(1)**	

In this input case, the number of power_iterations is 2, if it is 1 then return the initializing rank matrix or r(0). If iterations>2, the process repeats where you multiply the matrix, M with the new rank matrix r(t+1) at the next iteration.

Stepik Test Case Two Explanation:- (Power_iteration=3)

r(t+1)=r(1+1)=r(2) =M.r(1)=

	1	2	3	4	5			1			1
1	0	0	0	0	1/2		**1**	1/10	=	**1**	1/10
2	1/2	0	0	0	1/2		**2**	1/5		**2**	3/20
3	0	0	0	1	0	x	**3**	1/5		**3**	3/10
4	1/2	1	0	0	0		**4**	3/10		**4**	1/4
5	0	0	1	0	0		**5**	1/5		**5**	1/5
	M			**x**			**r(1)**		**=**	**r(2)**	

Template

You are allowed to use your own template but make sure your code passes the sample test cases. An example template to think about the problem is :

```
Class AdjacencyListorMatrix {
        private:
                //Think about what member variables you need to initialize
        public:
                //Think about what helper functions you will need in the algorithm
};

void AdjacencyListorMatrix::PageRank(int n){ }    // prints the PageRank of all pages after n powerIterations in
                                                  ascending alphabetical order of webpage and rounding rank to two decimal
                                                  places]

// This class and method are optional. To accept input, you can use this method:


int main()
{
   int no_of_lines, power_iterations;
   std::string from, to;
   std::cin >> no_of_lines;
   std::cin >> power_iterations;
   for(int i=0;i< no_of_lines;i++)
  {
        std::cin>>from;
        std::cin>>to;
        // Do Something
  }
  //Create a graph
  Created_Graph.PageRank(power_iterations);
}
```