ADS-2 Week 1 Exam - Section B

Instructions

1. Complete the Java program to pass the 12 test cases.
2. Detailed instructions on how to complete the program is given below.
3. Read it carefully.
4. Maximum marks 4 for test cases input000.txt to input010.txt
5. Maximum marks 1 for test case input011.txt

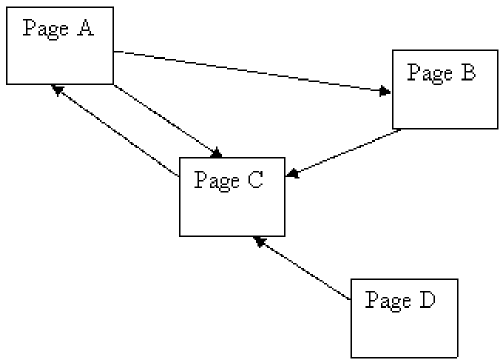
I'm Feeling Lucky!

Google has this "I'm feeling lucky" feature in the search engine. Type a search query in the Google search box and click on the I'm feeling lucky button. Google doesn't show you the search results. It directly takes you to the web page that is the best web page of all the web pages that are returned by the search engine. Try it!

We will attempt to write code such that the best web page is returned for a web search. To return the best web page you will implement the famous **PageRank** algorithm. BTW, PageRank is not web page rank, it is named after Larry Page, one of the co-founders of Google. PageRank (PR) is a way of awarding a score to a web page. **A higher PR would mean the web page is more important than the other web pages that have a lower PR**. In the next section let's understand how the PR score is computed.

Let's begin with how web search works. User types a query in the search box and the search engine gets all the web pages that match the query. For a query there could be several web pages that match and the goal is to find the best of these web pages. So, the **best web page is the one that has the max PR**.

Imagine the web as a **graph**. A web page is connected to other web pages with hyperlinks. A web page becomes a **vertex** and the link from one web page to the other web page becomes a **directed edge**. So, the PR of a web page is computed based on the PR of the incoming web pages pointing to it. It is an iterative formula and a worked out example is given below.



The figure above has 4 web pages. So, there are 4 vertices A, B, C and D. Let’s understand the edges.

1. Page A - 2 outgoing links pointing to Page B and Page C; 1 incoming link from Page C
2. Page B - 1 outgoing link pointing to Page C; 1 incoming link from Page A
3. Page C - 1 outgoing link pointing to Page A; 3 incoming links from Page A, Page B and Page C
4. Page D - 1 outgoing link pointing to Page C; 0 incoming links

Given this representation of the web pages with links pointing to each other, which web page is the most important? It is Page C as there are 3 incoming links. For a very large graph it is not practical to visualize and count the incoming links. To determine the importance of the page PR algorithm assigns a score for each web page. For the same example the PR for each web page is calculated as shown below.

PR(A) = PR(C) from previous iteration ÷ Outdegree(C)
PR(B) = PR(A) from previous iteration ÷ Outdegree(A)
PR(C) = PR(A) from previous iteration ÷ Outdegree(A) + PR(B) from previous iteration ÷ Outdegree(B) + PR(D) from previous iteration ÷ Outdegree(D)
PR(D) = 0 as there are no incoming web pages

Web Page	Incoming Web Pages	Outgoing Links	Iteration 0 PR (initial values)	Iteration 1	Iteration 2
A	C	2	1/4	¼ ÷ 1 = ¼	⅝ ÷ 1 = ⅝
B	A	1	1/4	¼ ÷ 2 = ⅛	¼ ÷ 2 = ⅛
C	A, B, D	1	1/4	(¼ ÷ 2) + (¼ ÷ 1) + (¼ ÷ 1) = ⅝	(¼ ÷ 2) + (⅛ ÷ 1) + (0 ÷ 1) = ¼
D	None	1	1/4	0	0

What are the initial PR values for the web page?
It is 1 divided by the total number of web pages.

When do we stop the iterations in the PR calculation?
In this program you may set the maximum iterations to 1000. Iterate 1000 times and finalize the values. Optionally, to optimize the running time of this loop, you may stop when the PR values don’t change after each iteration or after 1000 iterations.

Corner Case: What happens when there is a web page with 0 outdegree?
A web page with 0 outgoing links is called a dangling web page. This leads to a divide by 0 exception. The way to handle this exception is to add an edge from such 0 degree web page to all the other web pages in the graph. Test cases: input004.txt, input005.txt, input007.txt, input009.txt

Let's now see how to implement **PageRank** in Java

PageRank Class

1. Constructor - PageRank(Digraph) - The constructor should take the graph as input and complete the computation of the PR for all the web pages given in the graph.
2. double getPR(int v) - This method should return the final PR value (i.e., value obtained after 1000 iterations) of the vertex given as a parameter.
3. String toString() - return a string representation of PR for all vertices from 0 to v-1. The string format is v - PR\n.

You may add any other helper methods that you may require to complete the PR computation.

Solution class
This is the class with the main method and this method should handle the parsing of the test case input files. Let's understand the format of the input file.

Format of the test case input file

1. First line contains the number of vertices
2. From the second line to the count of vertices, each line represents the vertex in the first field followed by its adjacency list separated by space.

Format of the output

1. Print the graph in the adjacency list representation
2. Print the PageRank object (refer to the toString method for the print format)

WebSearch Class (Finish PageRank before you start this)
A file named **WebContent.txt** is provided to you. This file has 12 lines. Each line has two fields. First field is the web page vertex ID. Second field is the list of words that appear in the web page. These two fields are separated by a ":".

1. **Constructor** - WebSearch(PageRank, String filename) - Read the file WebContent.txt which is passed to filename parameter. Parse the file contents to build a hashtable. The words that appear in the web page are keys. For each key the value is the list of web page vertex IDs in which the word is present.
2. int **iAmFeelingLucky**(String query) - Lookup for the query in the hashtable. **If the query is not in the hashtable return -1**. If the query is in the hashtable then return the vertexID with the max(PR)

You may add any other helper methods that you may require to complete the web search.

Format of the input011.txt test case input file

1. First line contains the number of vertices
2. From the second line to the count of vertices, each line represents the vertex in the first field followed by its adjacency list separated by space.
3. List of queries, one per line that begins with “q=”.

Update the main method in Solution class to read the queries. For each query remove the prefix “q=” and pass the query to iAmFeelingLucky method. Print the value returned by the iAmFeelingLucky method to the output (one line for each query).