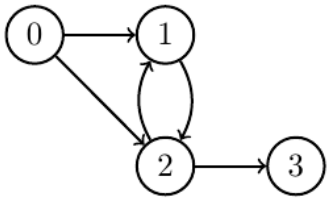
Lab 2 – Instructions

Some of our most used web services today are in different ways based on relations. They are used by Google to represent the link structure between web pages, and by Facebook to capture who knows whom. The purpose of this lab is to connect the abstract notions developed so far in the course with a small example that relates to a practical problem: ranking the importance of content on the Web.

1. Background

Graphs and relations

This image is a visual representation of a *graph*, where the bubbles are called *vertices* and the arrows are called *edges*. The arrows point out the direction of the edges, meaning, for example, that there is an edge from 0 to 1, but no edge in the other direction. Graphs whose edges have a direction are called directed graphs, and this is the only kind of graph we use in this lab.



In graph theory, a graph G is usually described by an ordered pair G = (V, E), where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, represented by a set of ordered pairs of vertices. Note that this means that E is a relation over V. Using this notation, the graph above is described as V = {0, 1, 2, 3} and E = {(0, 1), (0, 2), (1, 2), (2, 1), (2, 3)}.

PageRank and random surfers

The original idea behind Google is a graph-based Web page ranking called PageRank, where the vertices represent pages and the edges represent links. The Web is simply the collection of all links between pages. The page ranking algorithm is based on a model of a random surfer that randomly clicks on links to other pages, but sometimes gets bored and goes a random page on the whole web. The rank of a page is the probability of the random surfer ending up on that page.

In this lab, you will set up a tiny Web that has the structure of the figure above, and then estimate the PageRank for all the pages in that Web, by simulating a random surfer. The random surfer does the following on each step: with a probability of 1 − d, where d = 0.9, the surfer gets bored and surfs to a random page on the web, and with a probability of d, the random surfer clicks a random link on the current page, or, if there are no links, surfs to a random page on the web.

2. What to do

The logistics of this lab are similar to the previous one:

1. Download the skeleton project and unpack it.

2. cd into its top-level directory, `edaa40lab2`. Start the Leiningen REPL there.

3. Have a look at the file

```
edaa40lab2/src/edaa40/lab2.clj
```

This is the file you are supposed to edit. As in Lab 1, it contains commented-out incomplete code skeletons.

Your task is to implement all the definitions that have been commented out. Do not change the name or the parameter list, just add the function body or definition. Every commented-out function is preceded by a **declare** statement. Its purpose is to keep the compiler calm and allow you to load the package in spite of the fact that some functions initially remain undefined. If you try to call one of the declared-but-not-defined functions, you will get an error. After you have uncommented and implemented the function, you can remove the declare or just leave it there.

Right after some of the commented-out function skeletons are one or more **test?** statements. Their purpose is to help you check whether you are on the right track. Once you have an implementation, uncomment these tests, reload the package from the REPL (see below), and if the test passes (you will see something printed out on the REPL console) you might just have done it right. (As you can see, the tests are hardly exhaustive.)

From the Leiningen REPL you can load (and reload) the package using

```
(use 'edaa40.lab2 :reload)
```

while you work on the code and try it out.

Eventually, try running the page ranking algorithm by typing

```
(page-rank TinyWeb 100000)
```

into the REPL. The result should be something close to this:

```
{0 0.082061, 1 0.288512, 2 0.378093, 3 0.251334}
```

If you got there, show your work to the lab assistant.

PS:

This way of computing PageRank is not feasible for realistically sized graphs, but there are other ways of computing it. If you are interested, have a look at the Wikipedia page about PageRank:

`https://en.wikipedia.org/wiki/PageRank`