In this session:

- We will implement an efficient version of Pagerank on directed graphs.
- We will apply it to couple of examples, including a graph defined by airports and flights.

1 Pagerank

In class we have explained the iterative method for computing PageRank using matricial notation (i.e. the "power method"). The iterative step was the matrix-times-vector product $P(t) = M^T * P(t-1)$, but then we replaced M with the Google matrix $d * M + (1-d)/n * J$, where d is the *damping factor*. This was done in order to guarantee a unique solution and its fast convergence to it.

In this lab, we assume that a graph $G = (V, E)$ is given to us as a table of vertices + for each one, a list of successors. This is known as the *adjacency list* representation. In cases where we have significantly fewer edges than the maximum possible (e.g. in a graph with n nodes we may have $O(n)$ edges instead of the maximum $O(n^2)$ possible. In this case, the adjacency list representation is much more compact. Additionally for very large graphs we may not even be able to fit the full matrix into main memory, and would need instead to process few adjacency lists at a time.

An efficient implementation of the formula using an adjacency list representation of G instead of a matrix M would be:

```
 1. n = number of vertices in G
 2. P = any vector of length n and sum 1 (for example, the all 1/n vector)
 3. d = the chosen damping factor, between 0 and 1
 4. while not stopping condition:
 5.    for i in V:
 6.      L = [ j for (j,i) in E ]
 7.      Pnew(i) = (1-d)/n
 8.      for j in L:
 9.          Pnew(i) += d * P(j)/out(j)
10.    compute a distance between Pnew and P (to be used in stopping condition)
11.    P = Pnew
```

Silly question: How do we know that the division `/out(j)` does not give error?

The stopping condition we propose is that P and Pnew are sufficiently close under some distance you need to compute. "Close" is up to you: 0.01, 0.00001, 0.00000001, ... Try.

The damping factor d is of your choice. Popular ones are between 0.8 and 0.9. Something you can investigate for your report is how different choices affect the solution and how they affect the computation time.

Just to visually show how the algorithm works, you can play with an on-line implementation of pagerank here

2 Pagerank without inverting the graph

The scheme above is all fine for graphs that fit in RAM. But this need not be the case. In Big Data we will have graphs so large that at least the lists of edges have to be in disk. And, most likely, we will have a list of *outgoing* edges, not *ingoing* edges.

In other words, the algorithm above uses the list `[ j for (j,i) in E ]`. But if we are given G and not its inversion, what we will have instead is `[ j for (i,j) in E ]`.

Given this, we have two options:

- Invert the graph. This is similar to the problem of computing the inverted index. As it has very low disk locality if implemented naively, this takes time.
- Change the algorithm so that we compute pageranks incrementally. This is similar to what we did for computing tf-idf: invert the loops, and keep a set of partially computed tf-idfs (here, of partially computed pageranks). Similar to this:

```
 1. n = number of vertices in G
 2. P = any vector of length n and sum 1 (for example, the all 1/n vector)
 3. d = the chosen damping factor, between 0 and 1
 4. while not stopping condition:
 5.    Pnew = the all (1-d)/n vector
 6.    for i in V :
 7.        L = [ j for (i,j) in E ]. // forward adjacency list for node i
 8.        for j in L:
 9.            Pnew(j) += d * P(i)/out(i)
10.    compute a distance between Pnew and P
11.    P = Pnew
```

3 The input files

The following two files have been downloaded from Open Flights. A small donation has been made on your behalf.

`airports.txt` contains a list of airports from the world. The first fields are: an OpenFlights airport identifier, name of the airport, main city it serves, country, 3 letter IATA code, 4 letter ICAO code, and other stuff. (Only major airports have IATA codes). As an example, the first two lines of this file are as follows:

```
1,"Goroka","Goroka","Papua New Guinea","GKA","AYGA",-6.081689,145.391881,5282,10,"U"
2,"Madang","Madang","Papua New Guinea","MAG","AYMD",-5.207083,145.7887,20,10,"U"
```

`routes.txt` contains a list of routes from the world. The first fields are: an airline code, an OpenFlights airline code, an origin airport code (3 letter IATA code or 4 letter ICAO code), same with OpenFlight code, a destination airport code (3 letter IATA code or 4 letter ICAO code), then other stuff. In case of doubt about the file contents or decoding, go to OpenFlights page.

```
2B,410,AER,2965,ASF,2966,,0,CR2
2B,410,AER,2965,GOJ,4274,,0,CR2
```

Note that there is no guarantee that each airport mentioned in routes.txt has an entry in airports.txt. There may also be sinks, airports with some incoming route and no outgoing routes. *As you know, Pagerank needs to be patched to deal with sinks.*

We provide `read_airports` and `read_routes` functions to return a dictionary similar to `simple_graph`. It deals with sinks by simply removing them.

4 To do

1. Make sure you understand the first version of Pagerank above. Really, make sure, do not just transcribe it. Make sure you see the connection with the matrix formulation in the course slides.

2. Now go to the 2nd version (Big-data and disk friendly) of Pagerank. Make sure you understand the key difference, and why it should be better with data in disk. Write in the report why you think this is true, so that instructors see clearly that you have understood it.

3. Implement it (the 2nd version) and try it on the simple graph provided in the code.

   - Complete the `compute_pageranks(g)` function so that can it take the `simple_graph` variable as a parameter and return a dictionary of pairs (vertexname,pagerank).
   - Note that in the pseudocode P and Pnew are treated as vectors, e.g. assume that V is some set 1..n. In the real code, vertex names are strings so you have to turn them into dictionaries.
   - Check that the pagerank values you obtain are those stated in the course slide from which the example comes from. Test the effect of changing the damping factor.
   - If they don't come out right, first thing to check is that after each iteration the sum of P is 1 (or close to 1 except for tiny rounding errors). If it's not, you are doing something wrong in the iteration step.

4. Now try the piece of code at the end that computes the pagerank of airports and prints it in decreasing pagerank order. Are you surprised by the airports at the top?

5. Experiment with different values of the damping factor, and how it affects the convergence rate (iterations and time). Do your results seem consistent with the exponential convergence we saw in class?

6. **Bonus Track** (optional but recommended): The airport graph should be symmetrical or almost: if there is a route from A to B, it is highly likely that there is a route from B to A. It turns out that, for symmetric graphs, the pagerank has a very simple expression, computable by a very simple algorithm. Can you guess what it is? Can you prove your guess rigorously? Can you verify it on the airport graph? (you may want to try some simple examples first to make your guess).

7. Write the report explaining what you think you learned and what you found out. Do not paste code (unless you did something remarkable and unexpected).

5 Deliverables

Rules: Same rules as in previous labs apply.

To deliver: You must deliver 1) the program file or files you wrote, 2) a text file with the output of your program, and 3) a brief report (2 pages at the most) explaining the main difficulties/choices you had in implementing the scheme and any comments you have on the process or the result; this may include observations on the effect of the damping factor, closeness and speed of convergence, etc. Please do not deliver the `airports.txt` and `routes.txt` files, unless you've had to change them very substantially.

Procedure: Submit your work through the Racó as a single zipped file.

Deadline: Work must be delivered within **2 weeks** from the lab. Late deliveries risk being penalized or not accepted at all. If you anticipate problems with the deadline, tell me as soon as possible.