In this session:

- We'll use two text files to build a network of airports and flights
- We'll program a version of *Pagerank* on this data

1 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
```

In this lab we will work only with IATA codes and airports that have IATA codes. We will ignore the airports not having IATA codes, and also OpenFlight specific codes.

2 Pagerank

We want to compute the page rank of airports using the network defined by routes from and to airports. From the files above we can build a graph where the vertices are the IATA airports and the edges triples of the form (i, j, k) where i and j are IATA airports and k is the number of routes from i to j. Note that, unlike the version we explained in class, edges have weights, so the outdegree of i, $out(i)$, must be computed as the sum of all the weights of outgoing edges of i, that is, $out(i) = \sum_{(i,j,k)\in E} k$.

In class we have explained the iterative method for computing PageRank using matricial notation. An alternative notation for the same method is:

```
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. L = the chosen damping factor, between 0 and 1;
4. while (not stopping condition) {
5.    Q = the all-0 n-vector;
6.    for i in 0..n-1 {
7.      Q[i] =  L * sum { P[j] * w(j,i) / out(j) :
8.                        there is an edge (j,i) in G }
9.              + (1-L)/n;
```

```
10.     }
11.     P = Q;
12. }
```

The stopping condition may be a fixed number of iterations, or when two P's at consecutive iterations are sufficiently close (convergence). It is your job to think if this means 2 decimal places, or 4 decimal places, or 16 decimal places, or depending on n, or what.

The damping factor 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.

Among the lab files you will find a skeleton program `PageRank.py` that you may use as a basis.

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

3 To do

1. Make sure you understand the version of Pagerank above. Really, make sure, do not just transcribe it. You should be able to answer questions such as: what is `(1-L)/n` term in line 9 of the pseudocode mean? What is the role of `P` and `Q`?
2. Complete/modify the provided python program so it
   - reads `airports.txt` and `routes.txt` into appropriate data structures,
   - computes the page rank of every airport, and
   - writes down a list of pairs (page rank, airport name) ordered by decreasing page rank.

The crucial data structures you need will probably be:

- tables (arrays, hash, ...) linking airport codes, airport names, and indices as vertices.
- a graph, in a form that allows to retrieve efficiently all the triples (i, j, k) for a given i. It will be a bad idea (inefficient in time and memory) to build an $n \times n$ matrix, where n is the number of airports.

Important points:

- A basic check of your algorithm is that after each iteration the sum of elements in P is 1 (except for rounding errors). If it is not, something is wrong in your implementation.
- Efficiency of the solution is important. The time for an iteration must be linear in the number of edges. It is not ok to have memory or time always quadratic in the number of vertices. In this graph (and in most graphs for which we want to apply PageRank), the number of edges is $O(n)$, not $O(n^2)$. A necessary but not sufficient condition is that an execution on this graph should take seconds, not minutes.
- We do not guarantee that all airports in the first file have incoming and outgoing routes. Recall that these cases break down the Pagerank algorithm, as discussed in class. Take the necessary precautions in your implementation of PageRank. It is not ok to remove them: they have meaningful PageRank even with 0 outgoing edges, and it should be computed. Also, it is not ok to explicitly add all the edges from them to all others, as that will blowup the number of edges to n^2. Modify the pseudocode above to add the effect of these (virtual) edges efficiently: how many of them are there, and how much pagerank do they add in total to each vertex in particular?
- It may even happen that a few airport codes in the routes file do not appear in the airport file; it is ok to remove these if you want.

4 Deliverables

Rules: 1. You should solve the problem with one other person, we discourage solo projects, but if you are not able to find a partner it is ok. 2. No plagiarism; don't discuss your work with others, except your teammate; if in doubt about what is allowed, ask us. 3. If you feel you are spending much more time than the rest of the group, ask us for help. Questions can be asked either in person or by email, and you'll never be penalized by asking questions, no matter how stupid they look in retrospect.

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 raco platform 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.