Guillem Gili i Bueno

One of the things I wasn't quite sure how to do at first was how to change the base in function given in the session guide to something linear to the number of edges. Since I wasn't getting a good guess on it, I left if for when I had to implement the code, and by the time I reached it, it was easy to see that you just had to iterate over the edges and substract pagerank from the origin and add it to the destination.

When starting to code, I did and attempt to use the base code given, but I decided to change it a little bit. The main workflow of the program and functions stay the same, but I changed the data structures. I decided to keep only the necessary information, so I kept:
edgeHash which is the dictionary w(j,i) in the formula
airportHash which is the dictionary (IATA,airportname)
airportOutWeight which isthe dictionary out(j) in the formula
And that is it. The only other data structures in between were some dictionaries in order to store P and Q for the formula. I made sure initially each node had the same PageRank, and checked that after every iteration the sum of them is 1 (or something very close to 1) in order to ensure I was doing it correctly.
Also, since the damping factor is in every iteration and for every node the same I calculated it just once before executing the pageRank calculation. I included the files **airportst1** and **routes1** which was a very example to debug PageRank.

Overall, I'm proud of my code, executing 500 iterations(**outputcomp**) of Pagerank on the original set of airports and routes takes 20 seconds at most, what takes the most is loading the edges and vertices from the txt files, so it can do 25 iterations per second.

Testing

To have a referance of a "definitive" result I did a test with 100.000 iterations (**outputbase).** I compared it to the 500 iterations, and the difference is low enough to be dismissed.

To attemp to estimate how many iterations are needed, I added a function that tells you when all the values of the previous iteration varied so little it's not worth it to do it again, and experimented a bit with what value should I set as minimal difference.
My first guess was to put the (1 -L) /n of the formula, with an L of 0.9. I put it to the test against 500 iterations, and I observed that while it didn't perform flawlessly (had quite a few diferences) there weren't big diferences. For every airport I didn't see any case where it was more than 5 positions away than it should be, and the most top and most bottom results were accurate. It had around 50% of the document with small differences, then the other 50% was perfectly ordered. This was done in 25 iterations and took 6% of the time it took making 500 iterations, so it's a good tradeoff of speed vs accuracy This is in **outputcomp** and **outputcomp2.**

I also tried to change the damping factor to 1(**outputdamp** and **outputdamp2**), and got results a bit different since it's basicly nullifying it. First of all, the difference between

pageRank values is never less or equal to (1-L)/n, this makes sense because in this case it's 0. Second, we can see that the results are very different. This must be because one of damping factor's responsibilities is to nullify sinkholes, if we let it converge what we'll see is that only the nodes with no outcoming edges have a value different to 0. I tested it with 100.000 iterations, and while it's not 0, we can see a jump from a pageRank of 0.01 to 0.0006, which probably determines the line between the sinkholes(**outputdamp3**).