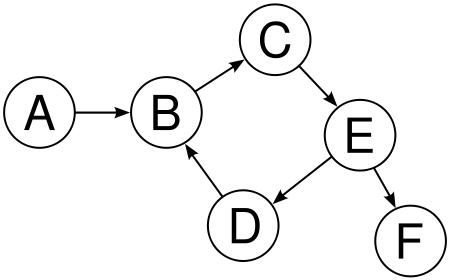
**DIRECTED GRAPH**

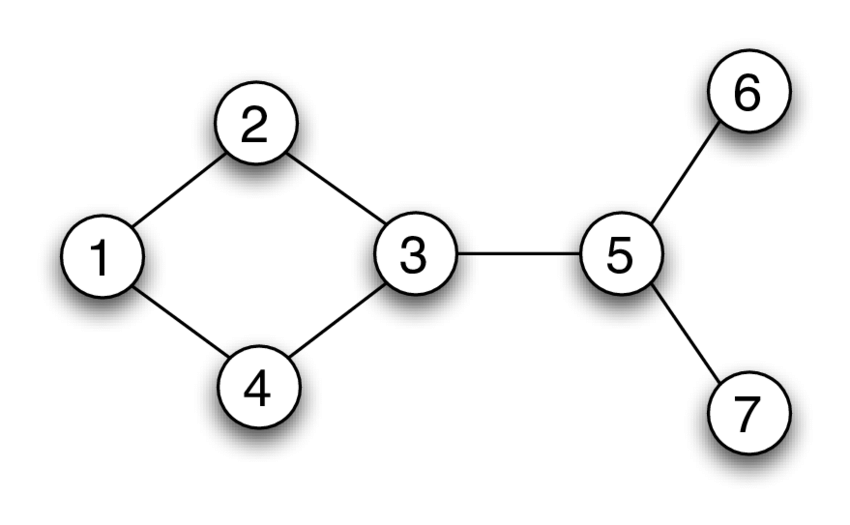


In this case the simple algorithm has been used. From reading the file it has been mapped and doubled with it’s inverse so:

Map(string s) { emit( [(split[0],[1,0]), (split[1],[0,1])] )

For out and in cases. Then the reducer just add by key. The average is then easy, cause the sum(number of them at the beginning) is given at hand. For a directed graph it is the same for in and out, so we just need to divide it by the number of elements after reducer.

**UNDIRECTED GRAPH**



In this case not the direction but the existence of an edge is important, therefore providing we use the same file, some cases can exist when we have the edge from the other direction after the inversion f.e:

(1->3), (4->1), (5->1)

So we can see that there is edge (3,1) that wasn’t included.

So we just inverse it and then we then need to remove all the duplicates f.e:

(1->3) and inverse (3->1)

The idea is:

* flatMap1(sting s) { emit((split[0], [split[1]]) , (split[1], [split[0]]) )

This function splits the directed input string the same way as before, so we can have duplicates. It vortex and it’s neighbour (whole edge)

* reduceByKey1(v,L[n]){ emit( list(set(a+b))) }

It reduces it the same way as for directed graph with exception that set removes duplicates with hashes!

* flatMap2(v, L[vi,vj,vk…]) { emit( [(v,v\_i),[dim(vi),L]…..

This map allows us to create emitter for each edge and owing to that we will then be able to create two of them. Therefore we will calculate two cases for an edge.

* reduceByKey2(e, [L1,L2]){ emit( e, [dimv1,dimv2, num of the same elems])}

This allows us to count how many the same elements are there exactly, so we have all needed information now.

* flatMap3(e,[dv1,dv2,n)){ emit [(v1,[c1,dimv1]),(v2,(c2,dimv2))]

We have calculated the coefficient and put it as a map

* reduceByKey(v,L[cv,dimv,1]){ emit((v,[sum(L),1])}

So now we have summed all up, and to calculate the average we would need to sum again by folder at the end