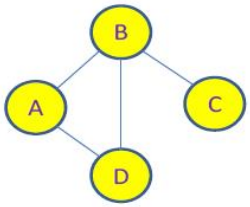
1. **[30]** Assume that we have a large undirected graph G(V,E), from which we can compute the neighbours $\rho(v)$ of a node $v \in V$ as the nodes adjacent to v in G. The common neighbours of (u,v) consist of those vertices in the intersection $\rho(u) \cap \rho(v)$.

(i) **[20]** Define the MapReduce pseudo-code for computing the common neighbours χ between every two vertices in a graph G.

(ii) **[10]** Apply your code to the example graph given below.



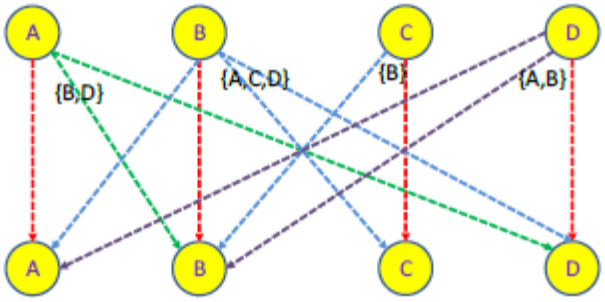
SOLUTION:

We will describe our Map and Reduce functions below, where we have a Mapper-CPU and a Reducer-CPU for every $v \in V$.

Map ($v \in V$)
  Do for $u \in \rho(v)$
    Emit {(v, u), $\rho(v)$}
  End do
End Map

\\ The reduce function for v gets as input $\{(v, u), \rho(v)\}$ and $\{(u, v), \rho(u)\}$ for all $u \neq v$

Reduce (v, $\{(v, u), \rho(v)\}$ and $\{(u, v), \rho(u)\}$)
  Do for every $w \in \rho(v)$
    Emit common-neighbour $\chi(v, w) = \rho(v) \cap \rho(w)$
  End do
End Reduce



χ(A,B): {B,D}$\cap${A,C,D}={D}
χ(A,D)={B,D}$\cap${A,B} = {B}

χ(B,C): {B}$\cap${A,C,D}={B}

χ(A,D)={B,D} $\cap$ {A,B} = {B}
χ(B,D)={A,C,D}$\cap${A,B} = {A}

χ(A,B): {B,D}$\cap${A,C,D}={D}
χ(B,C): {B}$\cap${A,C,D}={B}
χ(B,D)={A,C,D}$\cap${A,B} = {A}

(b) solution to provided graph

2. **[30]** We are interested in specifying how to process a simple web network using a MapReduce implementation of PageRank. Assume that we have the network shown in Figure 1.
    a. **[10]** Write out the pseudo-code for a MapReduce implementation of PageRank.
    b. **[10]** Apply the MapReduce algorithm to the network to compute the network weights, showing the weights at each step of MapReduce, for the first 3 steps of MapReduce.
    c. **[10]** Is MapReduce guaranteed to compute the same result as a centralised algorithm? Does it compute the same result in this example?

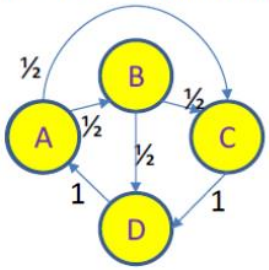
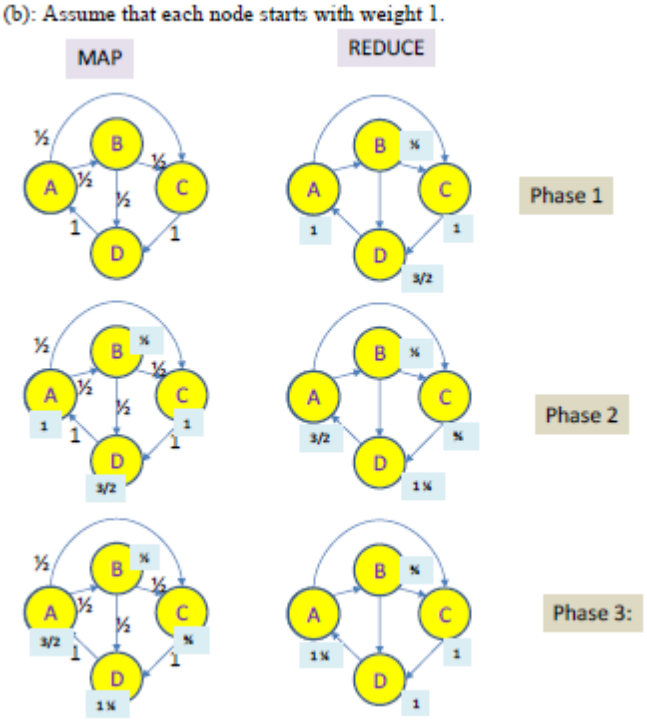
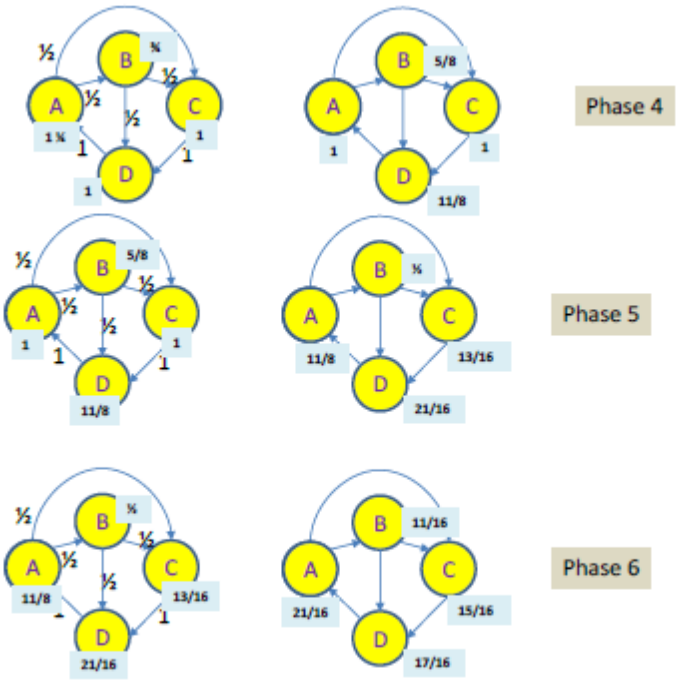


Figure 1: MapReduce network

Solution (a): see lecture notes
Solution (b): Assume that each node starts with weight 1.





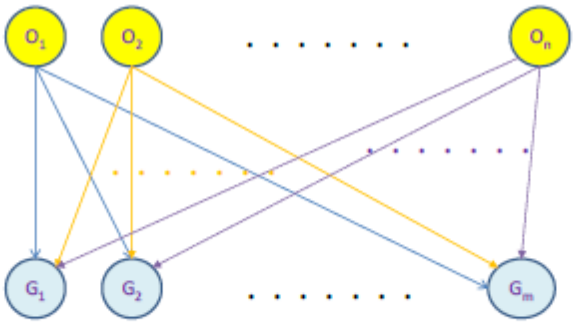
Solution (c): yes, iterative MapReduce is optimal for PageRank and computes the same value as a centralized version.

3. **[30]** The company Amazon wants to calculate the number of times customers ordered pairs of goods at the same time. For a set of goods $G=\{G_1, G_2, \ldots, G_m\}$, and a set of customer orders $O=\{O_1, O_2, .., O_n\}$, with $n>m$, Amazon wants to know the number of times customers ordered multiple items from G in an order in the set O of orders. We have up to n^2 CPUs available for processing at any time.

a. **[10]** Draw a figure showing the architecture of using MapReduce for solving the Amazon problem.

b. **[10]** Write out the pseudo-code for applying MapReduce to compute when customers ordered any two items together. In other words, we want to return the vector $\{(G_1, \Omega_1), (G_2, \Omega_2) \ldots, (G_m, \Omega_m)\}$, where Ω_i is the number of times good $G_i \in G$ was ordered with one other good.

c. **[10]** Extend this to compute the number of times a customer ordered *three or more goods* containing G_i (denoted $\#_i$), for each $G_i \in G$. In other words, we want to return the vector $\{(G_1, \#_1), (G_2, \#_2) \ldots, (G_m, \#_m)\}$.



We have a Mapper for every order, and a reducer for every good. Total CPUs is $mn < n^2$.

Solution (b):

MAP (O) \\ emits to reducer for G_j every time we find G_j in an order O_i with 2 or more items \\

If $|O_i|=k>1$, then do for j=1 to k

    Emit G_j such that $G_j \in O_i$

End

REDUCE (G_{j1}, …,G_{jk}) \\ Reducer for good G_j sums up the number of times a goods bundle has been ordered with at least 2 items containing G_j \\

Total ←0

Do for count = 1 to k

    Total ←total + 1

End

Emit total(G_j)

Solution (c):

We just need to **MAP** change the test for $|O_i|=k>1$ to $|O_i|=k>2$.