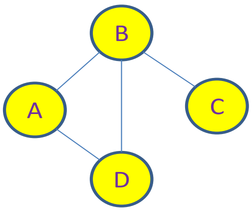
1. **[30]** Assume that we have a large undirected graph G(V,E), from which we can compute the neighbours $\rho(v)$ of a node $v \in V$ as the nodes adjacent to v in G. The common neighbours of (u,v) consist of those vertices in the intersection $\rho(u) \cap \rho(v)$.
   (i) **[20]** Define the MapReduce 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.



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?

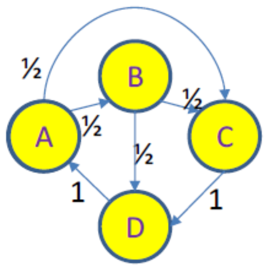


Figure 1: MapReduce network

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 $O=\{O_1, O_2, \ldots, O_n\}$, 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 $\{(G_1, \#_1), (G_2, \#_2) \ldots, (G_m, \#_m)\}$.