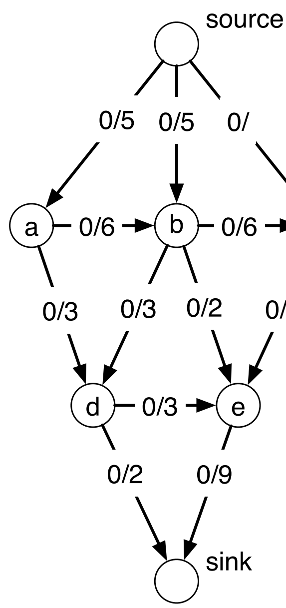
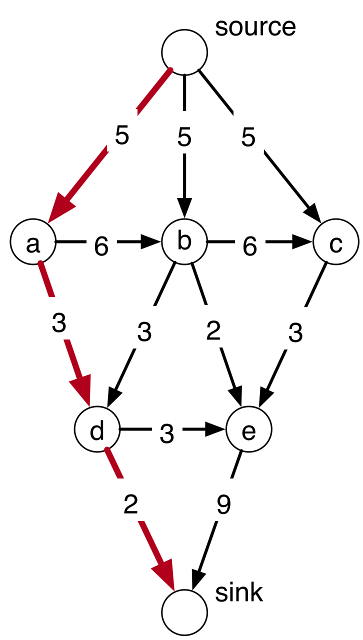
Homework 11 Solutions

Problem 1:

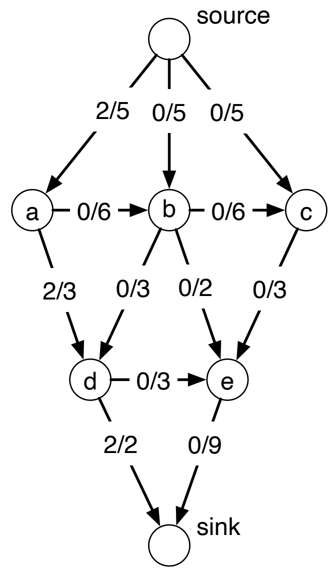
Step 1: We initialize the flow to be 0.



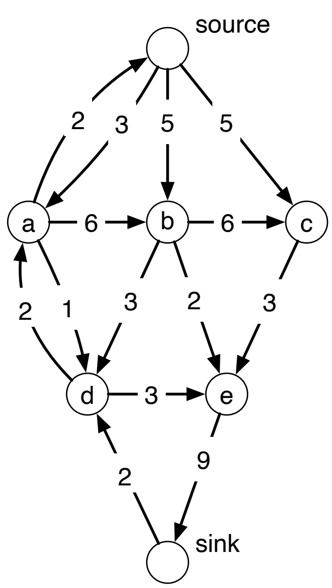
The residual has a path from source to sink:

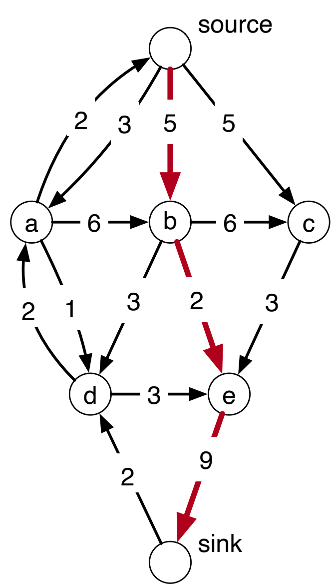


Step 2: The network flow augmented by the residual gives

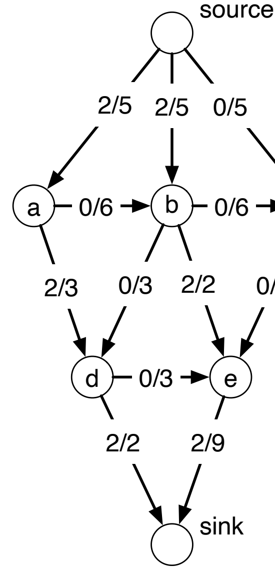


The residual graph is on the left. Since we are using BFS, the first path encountered is source-b-e-sink shown on the right. It gives a flow of 2.

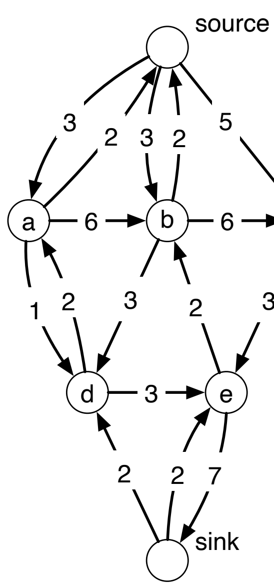


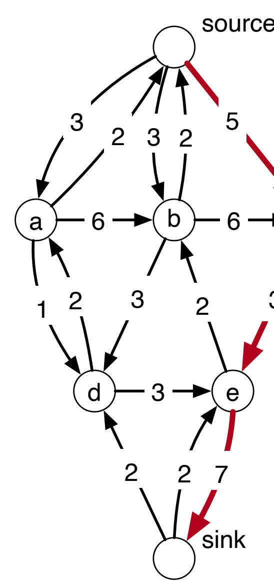


We augment and obtain

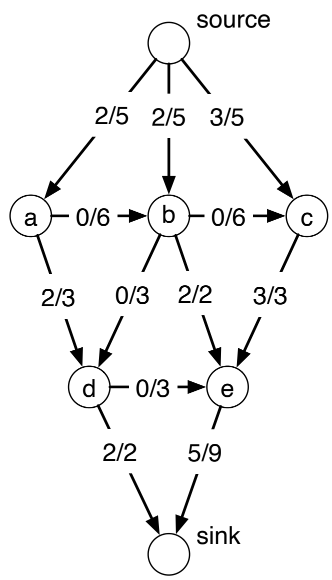


The residual graph is on the left. A BFS path is as short as possible, so we have no choice.

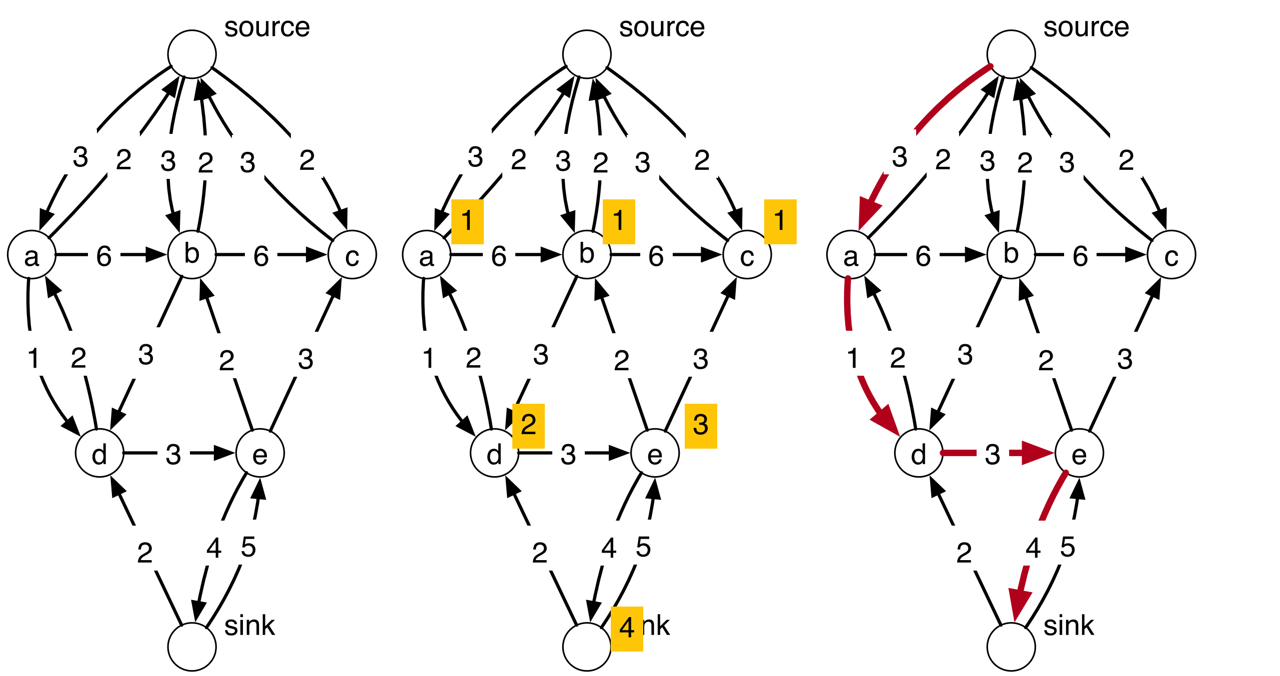




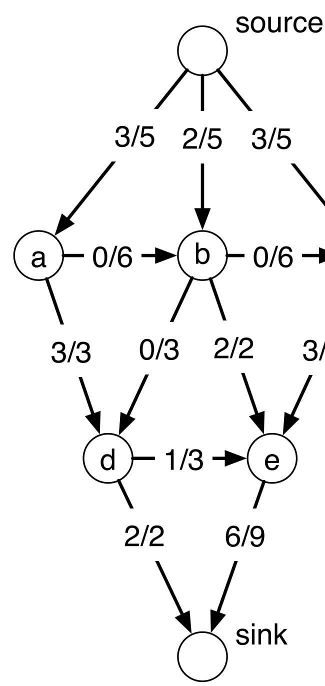
Incorporation into the flow graph gives a flow of 7



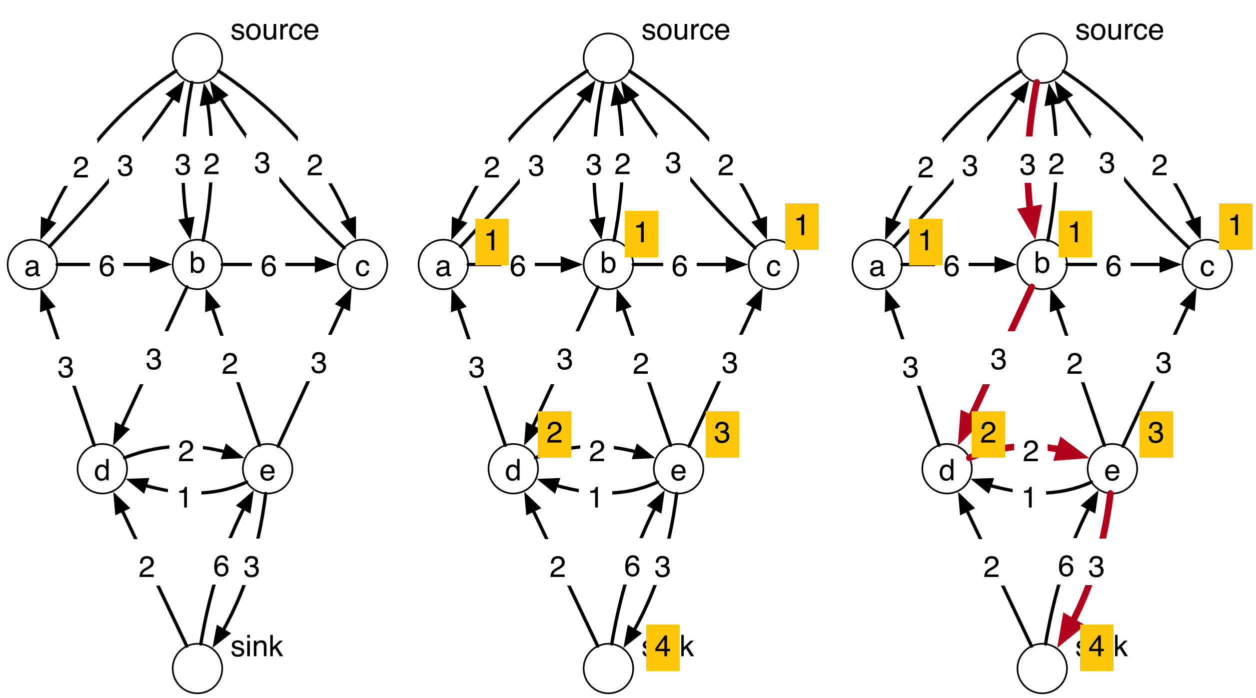
We now calculate the residual on the left and find a path with BFS. The yellow stickers give the distance from the source. As a result, we have an augmented flow of 1 on the right.



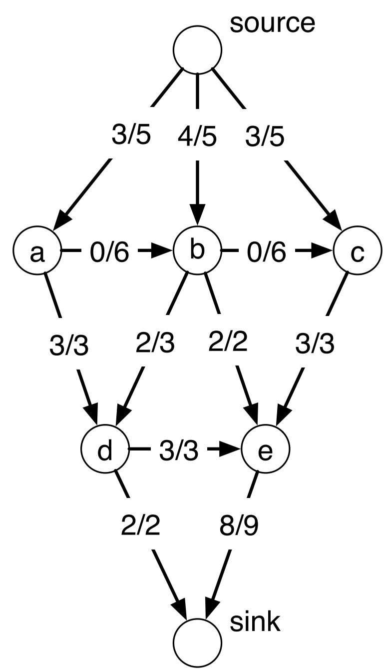
The resulting flow network is



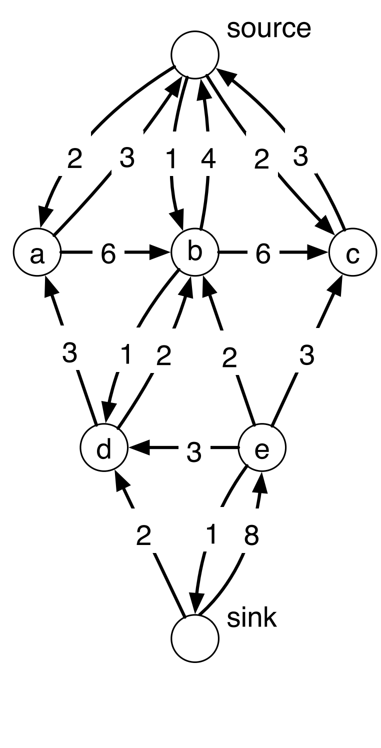
We calculate the residual, then do BFS with markers for the distance from the source, and finally obtain an augmenting path with a flow of 2.



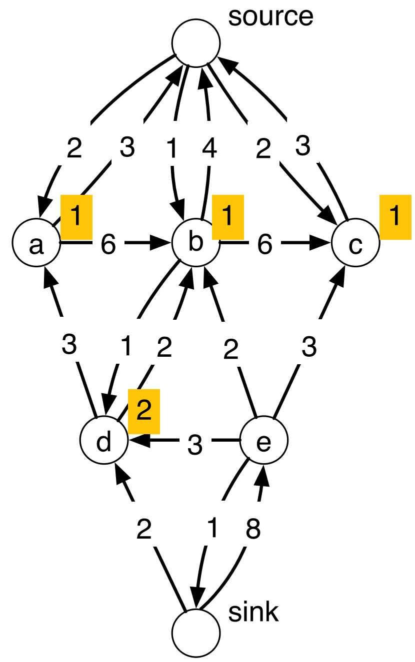
The resulting flow network is



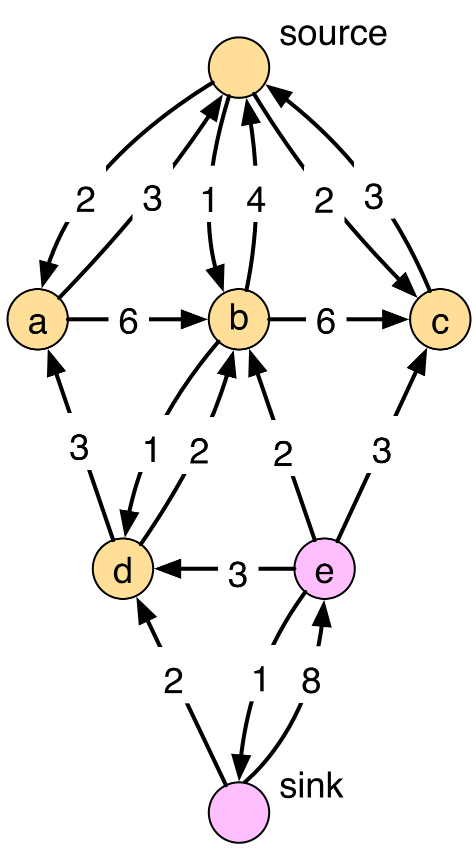
The residual is

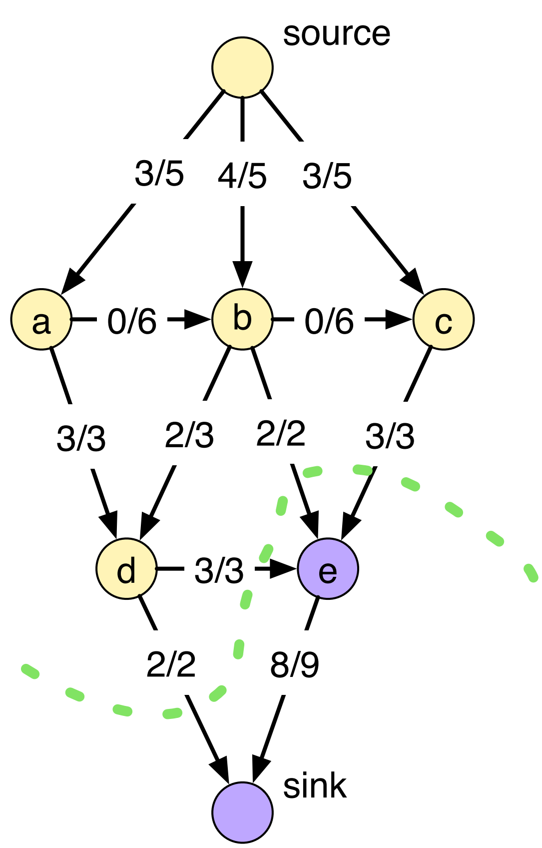


BFS gives the following distance from the source.



This shows that there is no longer a path in the residual from source to sink. We can put all reachable nodes into one set and the other ones into another set to obtain a cut. This gives a maximum flow of 10, realized in our flow.





Problem 2:

Let the array be a. Create a hash table for all the sums (e.g. using LH). The key is $a[i] + a[j]$ and the value is the pair of indices (i, j) with $i \leq j$. This will take $\frac{n(n+1)}{2}$ insertion, which we can assume to each take constant time. Then given c, we go through the array once more. For each index k, we look for $c - a[k]$ in the hash table. If we find it, then $c - a[k] = a[i] + a[j]$ for value (i, j). Thus, $c = a[i] + a[j] + a[k]$. The bill is $O(n^2)$ for creating the hash table and $O(n)$ for finding a triple sum, for a total of $O(n^2)$.

Problem 3:

Create a graph with vertices being the threads. Create an edge $t \rightarrow s$ if thread s waits for thread t. Then use DFS for a topological sort. If this works, then the threads are ordered in a manner where they can proceed. If this does not work, then there is a cycle. Progress is only possible if we break this cycle by randomly terminating a thread and thereby removing it from the graph. Since there are at most $\binom{n}{2}$ edges, the algorithm runs in time $O(n^2)$.