**Solutions to Practice Problems 7**

**Problem 1. Floyd-Warshall Recursion.**

The top-down Floyd-Warshall Algorithm is given as follows:

**Algorithm 1** Floyd-Warshall($w$)

```
1:  for i ← 1 to n do
2:      for j ← 1 to n do
3:          d[i, j] ← w[i, j]
4:          h[i, j] ← ∅
5:  for k ← 1 to n do
6:      for i ← 1 to n do
7:          for j ← 1 to n do
8:              newCost ← d[i, k] + d[k, j]
9:              if newCost < d[i, j] then
10:                 d[i, j] ← newCost
11:                 h[i, j] ← k
12: return d[ ], h[ ]
```

**Algorithm 2** get-path($i, j$)

```
1: if h[i, j] = ∅ then
2:     return (i, j)
3: else
4:     mid ← h[i, j]
5:     return get-path(i, mid) ⊕ get-path(mid, j)
```

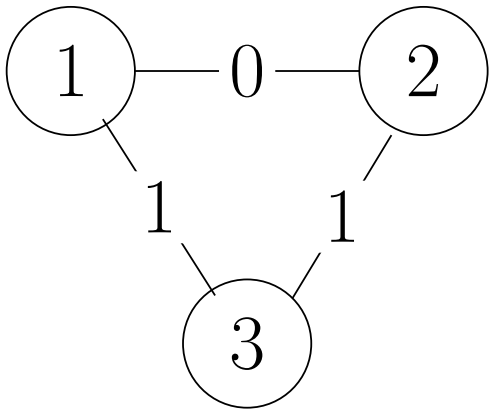
It was noted in the notes that if the "$<$" on line 9 of the Floyd-Warshall algorithm was changed to a $\leq$, the get-path function could cycle endlessly.

1. Explain what the $h[i, j]$ array represents in the output of the Floyd-Warshall algorithm.

   **Solution:** The $h[i, j]$ array gives the "hook" for a given path, which is used to obtain the final paths. The value is the index of the vertex through which the shortest path passes. The get-path function then obtains the vertex, and recursively checks the paths between that midpoint and merges them to obtain the shortest path.

2. Provide an example of a graph for which this change would result in infinite recursion in the get-path function. Demonstrate the Floyd-Warshall algorithm on this graph, and show how it results in an infinite recursion in the get-path function on some vertices $i, j$.

   **Solution:** Consider the following graph:




In the first step of the algorithm, the values of the arrays are:

| | $1,2$ | $2,1$ | $1,3$ | $3,1$ | $2,3$ | $3,2$ |
|---|---|---|---|---|---|---|
| d | 0 | 0 | 1 | 1 | 1 | 1 |
| h | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |

In the second step, for recalculating $d[1,3]$, the algorithm sees that passing through node 2 also costs 1. This sets $h[1,3] = 2$. However, it does the same for $d[2,3]$ and sets $h[2,3] = 1$.

| | $1,2$ | $2,1$ | $1,3$ | $3,1$ | $2,3$ | $3,2$ |
|---|---|---|---|---|---|---|
| d | 0 | 0 | 1 | 1 | 1 | 1 |
| h | $\emptyset$ | $\emptyset$ | 2 | 2 | 1 | 1 |

Then, calculating get-path between $1,2$ will recurse into get-path between $1,3$, which will recurse back into get-path $1,2$. This is an infinite loop that causes get-path not to terminate.

**Problem 2. Network Flows: Alternative Cases**

1. We consider an **Undirected Flow Network** to be an undirected graph $G = (V, E)$ with source and sink $s, t$ in which each undirected edge $e$ has a capacity $c(e)$. We let $G^D$ represent the **directed extension** of $G$, where for each edge $e = (u, v)$, $G^D$ instead has directed edges $(u, v)$ and $(v, u)$. A flow $f$ on an undirected $G$ is defined as a flow on the directed $G^D$.

   Show that if $G^D$ has a flow $f$ of value $k$, then there exists a flow $f'$ of value $k$ such that for each edge $e = (u, v)$, at least one of $f'(u, v)$ or $f'(v, u)$ is zero.

   **Hint:** Consider what it means for flow to be going both ways between two vertices

   **Solution:** Consider an undirected graph $G$, and a flow $f$ for the directed extension $G^D$ of $G$. We show a procedure that, for a pair of vertices $(u, v)$, creates a flow function $f'$ which only has flow on one direction of that pair.

   Suppose that $f(u, v) = a$ and $f(v, u) = b$. Without loss of generality, let $a \leq b$. We define $f'$ so that $f(u, v) = 0$ and $f(v, u) = b - a$.

   For vertex $u$, we have decreased the flow exiting $u$ by $a$, and decreased the flow entering $u$ by $a$. Thus, $u$ still has equal outgoing and incoming flow.

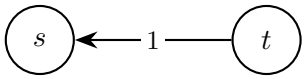
   We have also maintained $v$'s balance of outgoing and incoming flow by decreasing both by $a$.

Because $a \leq b$, the flow values of $f'$ are all nonnegative, as the new flow values are 0 and $b - a$.

This procedure may be applied repeatedly on $f$ to create a new flow in which every pair of directed edges between two vertices never both have positive flow.

2. Give an example of a directed graph $G = (V, E)$, such that if $G$ was made into an undirected graph $G'$, the maximum $s, t$ flow of $G'^D$ would be greater than that of $G$.

   **Solution:** The extremely simple solution to this problem is a graph with one directed edge from $t$ to $s$ as follows:




   No flow is possible in the directed version, but the undirected flow is 1 as we may pass from $s$ to $t$.

3. Suppose you are given a device which is able to find the maximum flow for a graph $G = (V, E)$ with source and sink $s, t$. However, this device requires that $c(e) = 1$ for all edges. Come up with an efficient procedure so that you could use this device to find maximum $s, t$ flow for any graph.

   **Solution:**

   The idea here is that the "capacity" of a directed edge can be mimicked by using several directed edges of lesser capacity. For example, if $c(e) = 2$, then two duplicate copies of $e$ with capacity 1 would recover the same flow. Thus, we will create a graph $G'$ such that for each edge $e \in E(G)$, $G'$ has edges $e_1, \cdots, e_{c(e)}$.

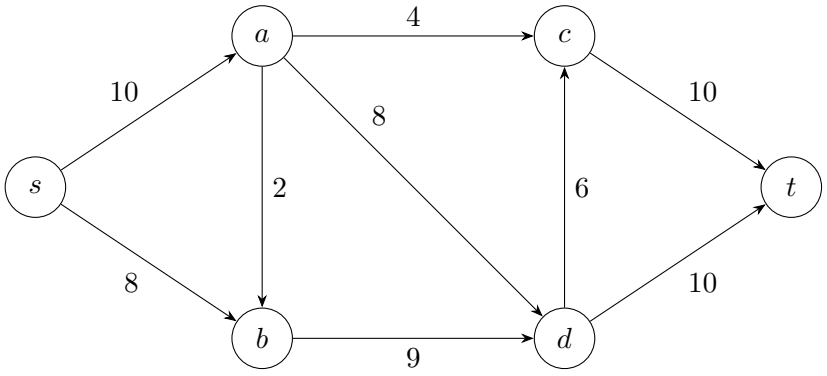
   For a flow on $G'$, we may convert to a flow on $G$ by putting flow through $e$ equal to the number of its corresponding capacity-1 edges in $G'$ that received flow.

   Similarly, a flow in $G$ may be converted to a flow in $G'$ by adding 1 flow to some number of corresponding edges equal to the flow through every directed edge in $G$.

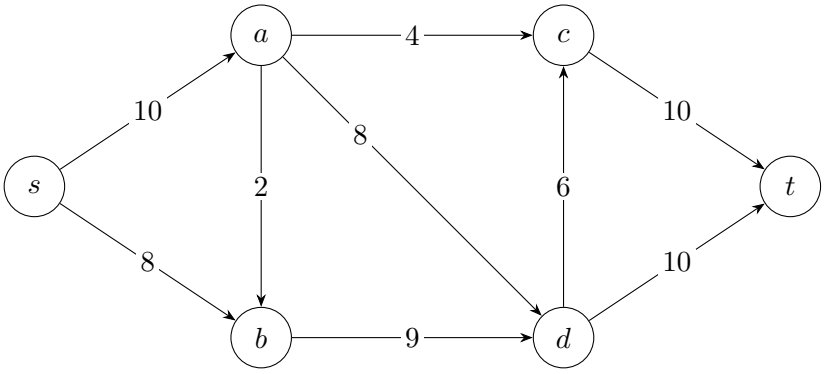
   Thus, our device that obtains the maximum $s, t$ flow for graphs with maximum capacity 1 may be used to obtain the maximum $s, t$ flow for graphs with arbitrary maximum capacity by reducing the arbitrary graphs into graphs $G'$ with large numbers of parallel edges.

**Problem 3. Network Flows: A Worked Example:**

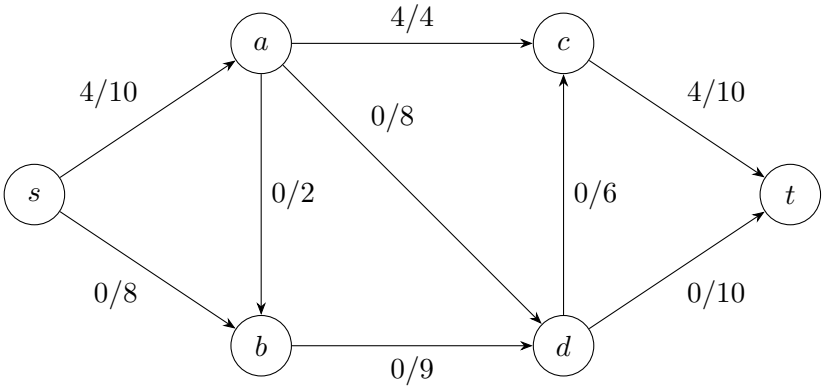
Demonstrate the Ford-Fulkerson Algorithm as given in class on the following graph, and determine the flow function $f$ which obtains the optimal $s, t$ flow.



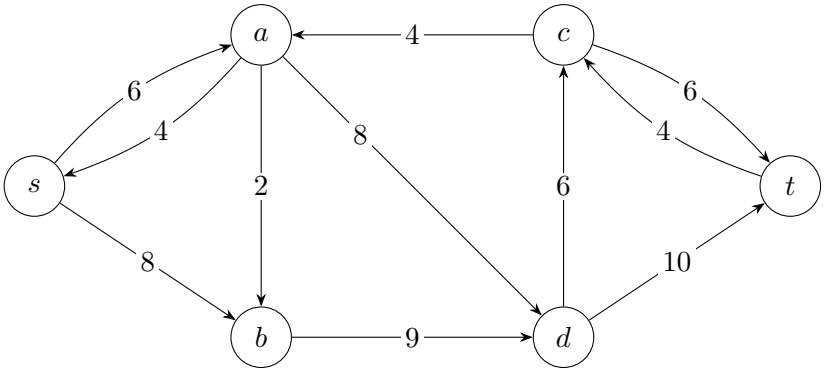
**Solution:** Initial residual graph:



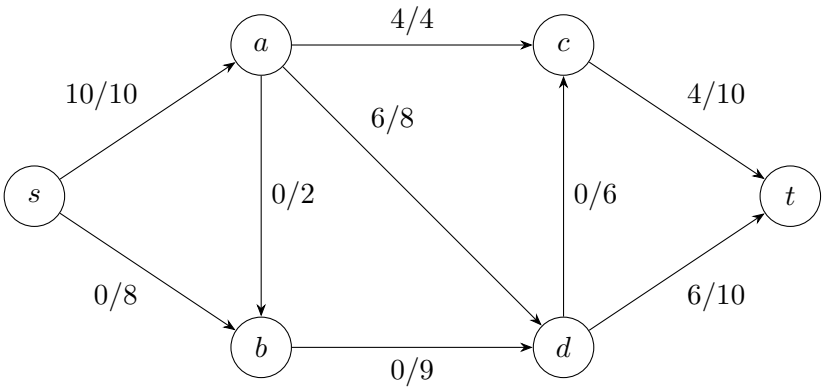

First flow (path from $s \to a \to c \to t$)



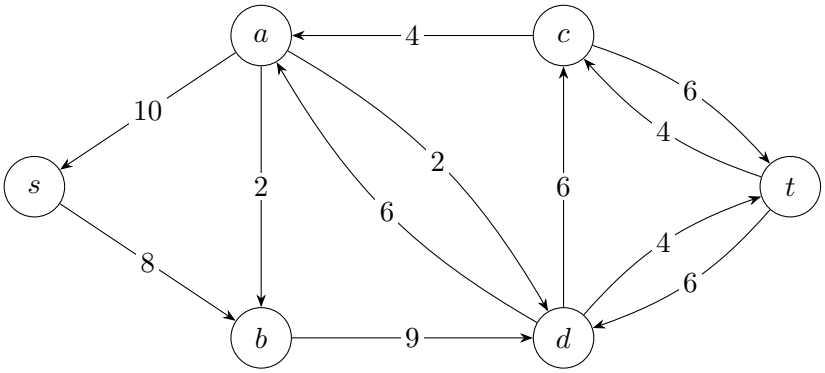

Second residual graph:



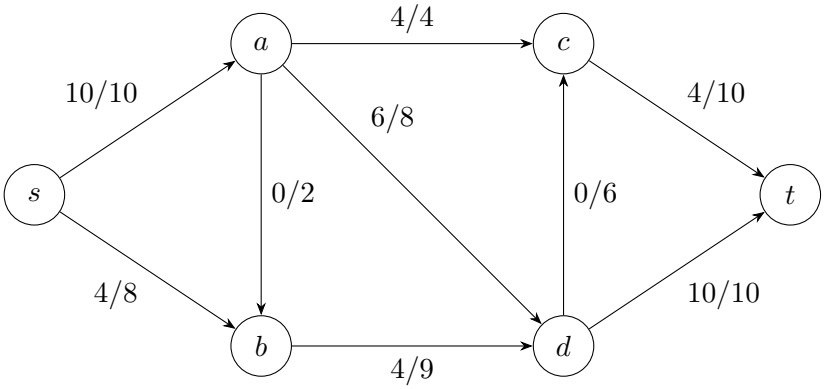

Second flow (added path $s \to a \to d \to t$):



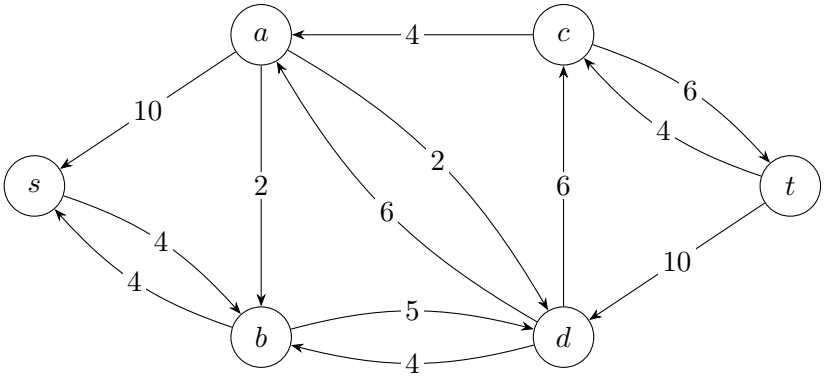

Third residual graph:




Third flow (path from $s \to b \to d \to t$:

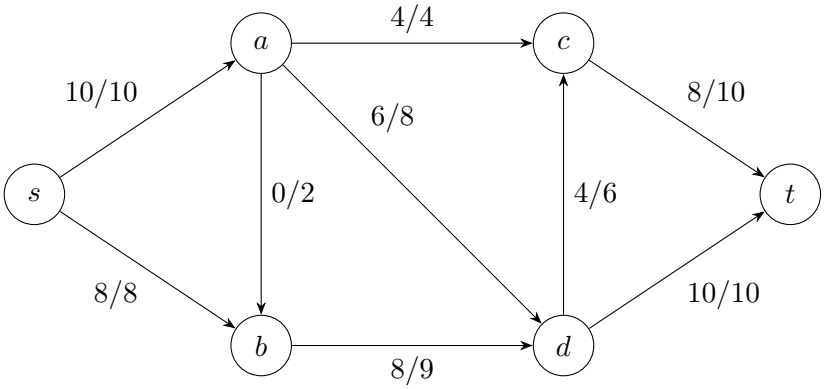



Fourth residual graph:

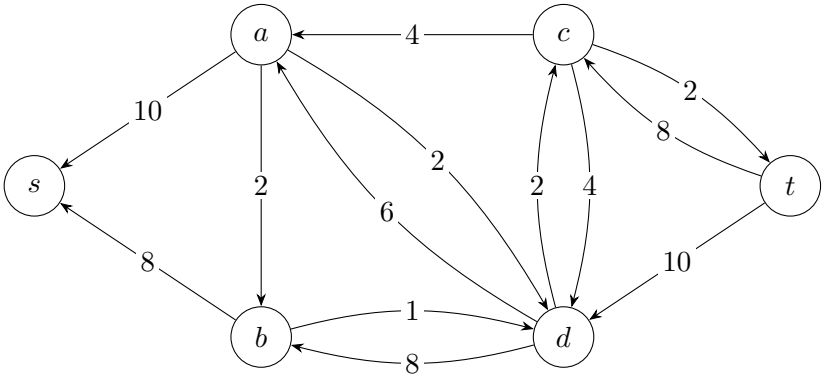



Fourth flow (path from $s \to b \to d \to t$):

**Flow graph 4** $(|f| = 18)$




Final residual graph:




There is no $s, t$ path in this residual graph, so the flow is maximum at 18.

**Problem 4. Increased Capacity**

Suppose you are given a graph $G = (V, E)$ with capacity function $c$, source and sink $s, t$, and maximum $s, t$ flow $k$. You increase the capacity of a single edge $e$ by $r$, creating new capacity function $c'$. Show that the maximum $s, t$ flow under $c'$ can be at most $k + r$.

**Hint: Consider the how this affects the residual graph for the initial maximum flow.**

**Solution:**

Recall that a maximum flow is found when there is no $s, t$ path in the residual graph. Consider the original maximum flow $f$ for the graph $G$. In the residual graph, there must therefore be no $s, t$ path.

If we consider the residual graph after the capacity is updated to add $r$ to $c(e)$, there may still be no $s, t$ path. However, if there is, it means that:

- The flow over $e$ was previously maximized.

- There exists a cut in the residual graph over which the only edge crossing in the $s \to t$ direction has capacity $r$.

Thus, the maximum amount of additional flow that can be added to this graph is $r$, and the flow may increase by at most $r$.

Additionally, one may show by contradiction that if a flow of value $> r + k$ existed in the modified graph, then one could relax flow along modified edge $e$ to bring its flow down by $r$ and result in a flow valid in the original graph of value $> k$, contradicting its optimality.