15-451 Algorithms, Spring 2016
Recitation #7 Worksheet

Zero-Sum Games and Randomized Lower Bounds

Solving for minimax optimality: Here is a version of the shooter-goalie game. Shooter is the row player, Goalie is the column player. The matrix entry is the payoff to the shooter (row player) if the shooter and goalie each choose the corresponding action (think of it as the chance the ball goes in). So, the row player wants to maximize the expected payoff and the column player wants to minimize it.

	Left	*Right*
Left	1/2	1
Right	1	1/4

What are the minimax optimal strategies and what is the minimax value of the game (to the shooter)?

Solution: We can solve for the shooter's minimax optimal strategy as follows. Say the shooter puts probability p on shooting left and $1-p$ on shooting right. Her payoff if the goalie dives left is $p/2+(1-p)=1-p/2$. Her payoff if the goalie dives right is $p+(1-p)/4=3p/4+1/4$. We can maximize the minimum by setting them equal (since one is increasing with p and one is decreasing with p). This gives us $1-p/2=3p/4+1/4$ so $3/4=5p/4$ so $p=3/5$. The value to the shooter is $7/10$. We can analyze the goalie the same way. Say the goalie puts probability q on diving left and $1-q$ on diving right. This gives shooter a payoff of $q/2+(1-q)=1-q/2$ for shooting left, and $q+(1-q)/4=1/4+3q/4$ for shooting right. Goalie wants to minimize shooter's payoffs, which can be done by equalizing them (since one is increasing with q and one is decreasing), solving to $q=3/5$. Note that this guarantees the shooter does no better than $7/10$. So, the value of the game is $7/10$.

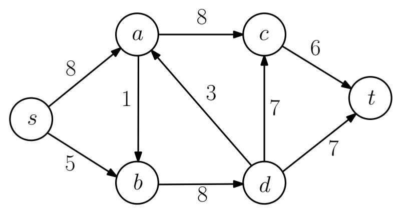
Lower bounds for randomized algorithms: Go through the analysis from the lecture notes for how to give a lower bound for *randomized* comparison-based sorting algorithms. This is in the notes but we didn't do many details in lecture saying that we would go through it in recitation. Here are the bullet points:

- First of all, in a zero-sum game, if there is a randomized strategy for the column player that guarantees the row-player pays at least V in expectation no matter what row it plays, this implies the row-player pays at least V in expectation even if it uses a randomized strategy.
- This implies that if there is a probability distribution over inputs such that any *deterministic* algorithm has expected cost at least V, then any *randomized* algorithm also has expected cost at least V. (This is called "Yao's principle").

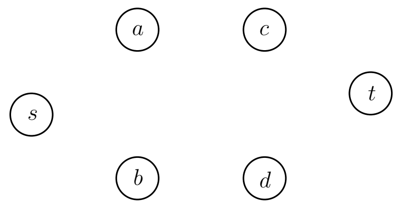
- In the case of comparison-based sorting, we can choose the uniform distribution over all $n!$ permutations of $\{1, \ldots, n\}$ as the randomized strategy for the adversary / column-player.
- Any deterministic sorting algorithm can be viewed as a binary tree, with each internal node having a comparison, and with $n!$ leaves, one for each ordering of the input elements (since we know the algorithm cannot do the same thing on two different input orderings if it wants to be correct, since the input elements are all distinct).
- The number of leaves at depth $\leq \lg(n!) - 10$ is at most $1 + 2 + 4 + 8 + \ldots + n!/1024 < n!/512$. This means that over 99% of the leaves are at depth $\geq \lg(n!) - 10$, which implies the average depth is $\Omega(n \log n)$.
- Since the average depth is $\Omega(n \log n)$, this means the algorithm pays $\Omega(n \log n)$ in expectation to sort a random ordering of $\{1, \ldots, n\}$.
- By Yao's principle, this then implies that even a *randomized* comparison-based sorting algorithm cannot beat the $\Omega(n \log n)$ lower bound.

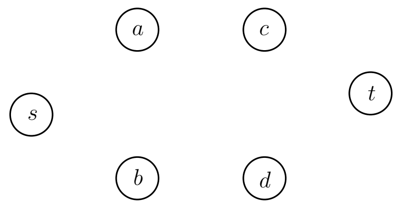
Network Flows.

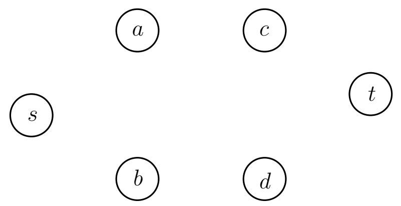
Example of running Ford-Fulkerson: Here is a problem from the midterm in 2013. Consider the graph below.

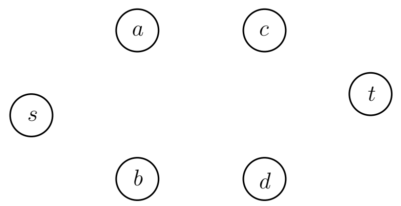


Run Ford-Fulkerson and show the final residual graph. Also what is the maximum flow?









Solution: Max flow is 12.