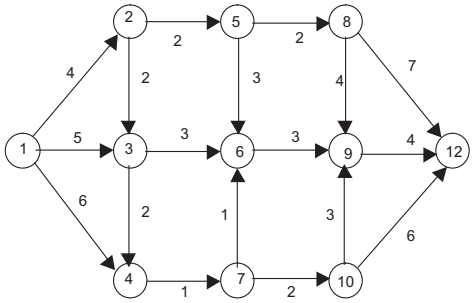
EC 504
Spring, 2021
HW 6

Due Wednesday, April 7, 8PM on Gradescope.



1. (15 pts) Consider the graph in the figure above as a directed, capacitated graph, where the numbers indicate an arc's capacity to carry flow from node 1 to node 12. In the max-flow algorithm of Ford and Fulkerson, the key step is, once a path has been found, to augment the flow and construct the residual graph for the next iteration.

   (a) Consider the path $1 \to 2 \to 3 \to 6 \to 9 \to 12$. What is the capacity of this path?
   Solution: It is the minimum of the capacities of the arcs which is 2.

   (b) Suppose we send two units of the flow along path $1 \to 2 \to 5 \to 8 \to 12$. Construct the residual graph which remains after this flow has been sent. Draw this graph.
   Solution: The changes are as follows: The capacity of the arc (1,2) is reduced by 2 to 4. A reverse arc (2,1) is introduced, capacity 2. The arc (2,5) is removed and a reverse arc (5,2), capacity 2 is introduced. The arc (5,8) is removed, and a reverse arc (8,5), capacity 2 is introduced. The arc (8,12) has reduced capacity 5, and a reverse arc, (12,8) is introduced with capacity 2. The rest of the graph does not change.

   (c) Find the maximum flow from node 1 to node 12 in this graph.
   Solution: This requires working through the Ford-Fulkerson algorithm. Total flow will be 6. You can place 2 units along 1-2-5-8-12, 3 units along 1-3-6-9-12, and 1 unit along 1-4-7-10-12. Those are all minimum hop paths. Then, there are no paths left, because one cannot cross from 1-2-3-4 to the rest of the nodes. Alternatively, note the cut of capacity 6, obtained by removing arcs 2-5, 3-6, 4-7.

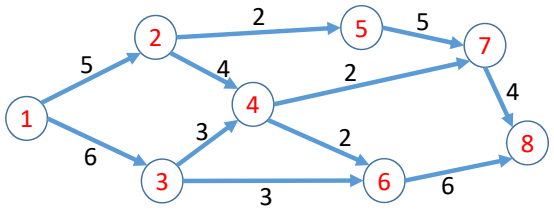
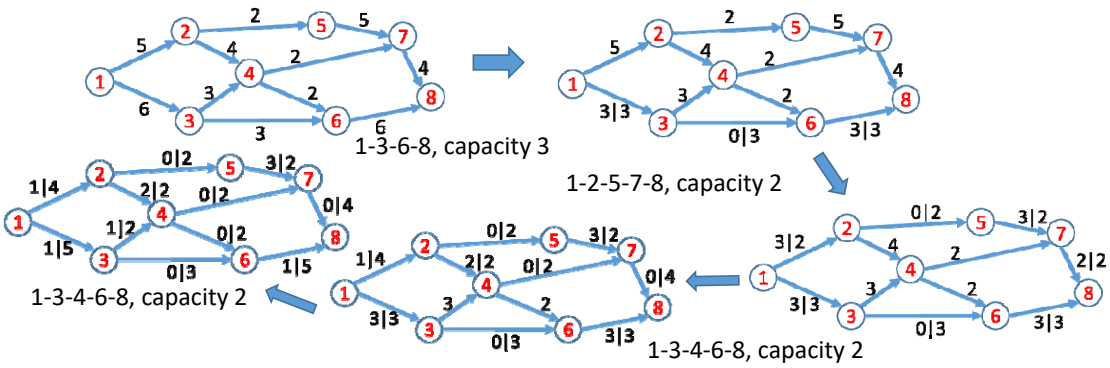


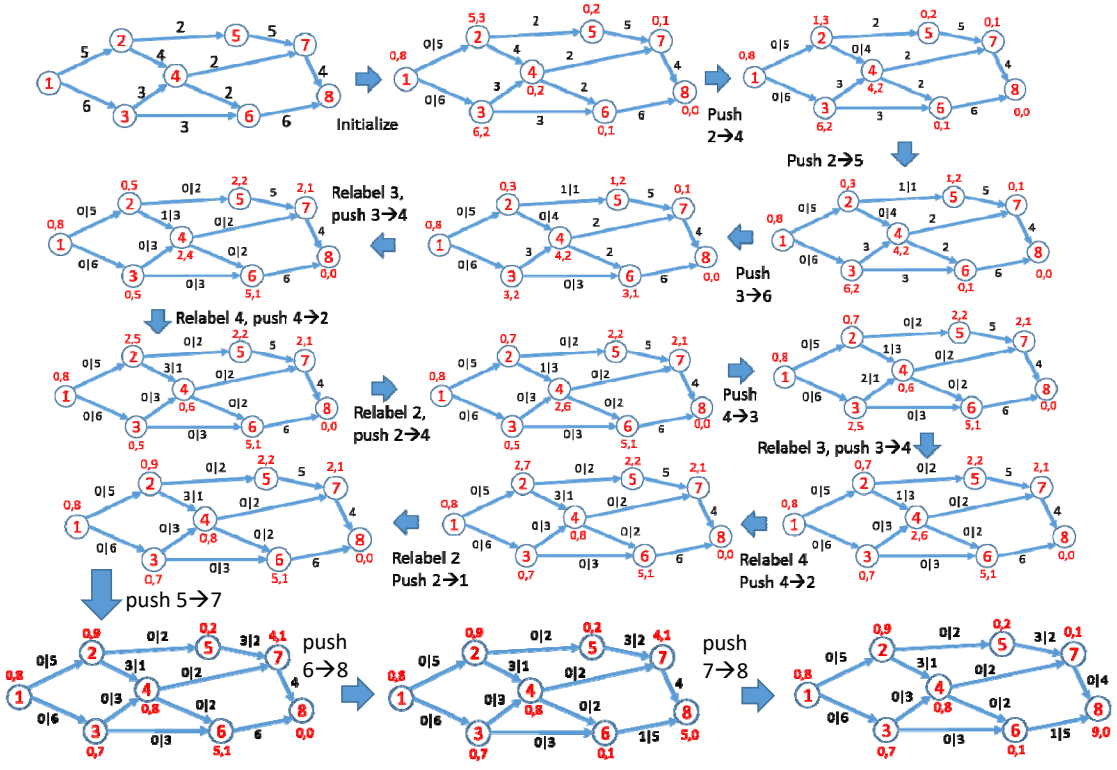
Figure 1: Figure for Problems 2, 3.

2. (15 pts) Consider the weighted, directed graph in Figure **??**. Illustrate the steps of the Ford-Fulkerson algorithm for finding the max flow from node 1 to node 8. Show the residual network at each step.
   Solution: We will do this using breath-first search for paths. The steps are shown in the figures below. We use two numbers per arc to indicate the forward capacity — backward capacity in the residual network.



3. (15 pts) Consider the weighted, directed graph in Figure 2. Illustrate the steps of the preflow-push algorithm for finding the max flow from node 1 to node 8. Show the distance labels of each node at each step, and the excesses.

Solution: We show the excesses and the distance labels as an ordered pair in red in the algorith illustrations below. The residual graph capacities are shown in black as a pair also. Note that, although there are more iterations here than in Ford Fulkerson, the complexity of each iteration is much smaller. A single Ford Fulkerson iteration requires a full breadth first search, which roughly equals 6 preflow-push iterations in this graph!



4. (15 pts) Let $G = (V, E)$ be a directed, capacitated graph with source s, t and integer capacities. Suppose that we are given a maximum flow $f(u, v), (u, v) \in E$, in the network G.

   (a) Suppose that we increase the capacity of a single edge $(u', v') \in E$ by one unit. Describe an algorithm of

$O(|V|+|E|)$ to modify the previous max flow $f(u,v),(u,v) \in E$, to a new max flow in the new network where the edge (u',v') has increased capacity by one.

Solution: If the current flow does not have $f(u',v') = c(u',v')$, then there increasing the capacity $c(u',v') = c(u',v') + 1$ won't matter, as this won't create an augmenting path in the residual network. Otherwise, construct the residual network R_f with the increased capacity edge $c(u',v') = c(u',v') + 1$. Run one iteration of Edmonds-Karp using BFS to search for an augmenting path in the network R_f. It runs in $O(|V|+|E|)$, and it may find an augmenting path of capacity 1, or not. In either case, the new max flow is determined.

(b) Suppose that we decrease the capacity of a single edge $(u',v') \in E$ by one unit. Describe an algorithm of $O(|V|+|E|)$ to modify the previous max flow $f(u,v),(u,v) \in E$, to a new max flow in the new network where the edge (u',v') has decreased capacity by one.

Solution:

If $f(u',v') < c(u',v')$, then the max flow will not change if we decrease the capacity $c(u',v') = c(u',v')-1$. Otherwise, find a path from s to t in using only edges in E which have positive flow on the edges. This can be done with BFS in $O(|V|+|E|)$. Decrease the flow on this path by 1, to define new flow $f'(u,v)$. Construct the residual network $R_{f'}$, and try to find an augmenting path in $R_{f'}$. This takes $O(|V|+|E|)$. If you find an augmenting path, perform the augmentation, which leads to a new flow. If no augmenting path is found, the flow f' is optimal.

5. (10 pts) Assume you have a scheduling problem with 10 customers. Customer i has value V_i, and requires processing time T_i. The values of V_i and T_i are listed below as arrays:

```
V = {3,4,7,5,2,3,5,8,9,6}
T = {2,3,4,3,1,3,3,5,6,4}
```

Assume that jobs can be scheduled partially, so that a job of value V_i which requires time T_i will receive value $V_i t/T_i$ if processed only for time T.

Find the maximum value which can be scheduled with total processing time 10 units.

Solution: This is a fractional knapsack problem, since you get partial credit. Compute the ratios of value per time to obtain: sorted according to value per unit capacity, which is:

$$V/T = \{1.5, 1.33, 1.75, 1.67, 2, 1, 1.67, 1.6, 1.5, 1.5\}$$

Thus, the task order is $(5,3,4,7,8,1,9,10,2,6)$.

Using the greedy algorithm, we add task 5 at capacity 1, task 3 at 4, task 4 at 3 to use up 8 units. Task 7 only gets 2 units, so has value 2/3 of 5. Total value is $2+7+5+10/3 = 171/3$.

6. (10 pts) Assume you have a scheduling problem with 10 customers. Customer i has value V_i, and requires processing time T_i. The values of V_i and T_i are listed below as arrays:

```
V = {3,4,7,5,2,3,5,8,9,6}
T = {2,3,4,3,1,3,3,5,6,4}
```

Assume that jobs must be scheduled fully, with no partial credit.

Use dynamic programming to find the optimal value which can be scheduled in a total of 9, 10, 11, 12 time units. (Note: this may be easier to do in MATLAB or Python than by hand.)

Solution: This is an integer knapsack problem, which we solve with dynamic programming. The optimal value function is a function $J(i,C)$, where i is the largest integer for the tasks that are being considered and C is the available capacity. We use the relationship:

$$J(i,C) = min\{J(i-1,C), V_i + J(i-1, C-T_i)\}$$

The table is shown below:

\ c = 0	1	2	3	4	5	6	7	8	9	10	11	12
i:												
1	0	0	3	3	3	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7	7	7	7
3	0	0	3	4	7	7	10	11	11	14	14	14
4	0	0	3	5	7	8	10	12	12	15	16	16
5	0	2	3	5	7	9	10	12	14	15	17	18
6	0	2	3	5	7	9	10	12	14	15	17	18
7	0	2	3	5	7	9	10	12	14	15	17	19
8	0	2	3	5	7	9	10	12	14	15	17	19
9	0	2	3	5	7	9	10	12	14	15	17	19
10	0	2	3	5	7	9	10	12	14	15	17	19

From the table, the optimal values for capacities 9, 10, 11, 12 are 15, 17, 19 and 20.

The optimal assignments are: For capacity 9, the tasks scheduled are: 1, 3, 4.

For capacity 10: the tasks scheduled are: 1,3, 4, 5.

For capacity 11: the tasks scheduled are: 3, 4, 5, 7.

For capacity 12: the tasks scheduled are: 1,3,4,7.