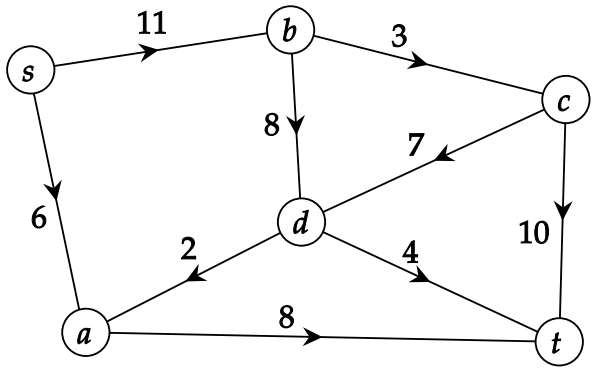
CMPSC 465 Data Structures & Algorithms
Fall 2024 Mingfu Shao and Debarati Das

# HW 7

## Due October 18th (Friday), 11:59 pm

Formatting: Each problem should begin on a new page. When submitting in Gradescope, try to assign pages to problems from the rubric as much as you can. Make sure you write all your group members' names. For the full policy on assignments, consult the syllabus.

1. (6 + 3 + 3 pts.) Consider the following network $G = (V, E)$ with source $s$ and sink $t$:



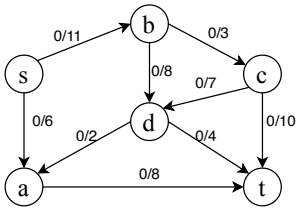
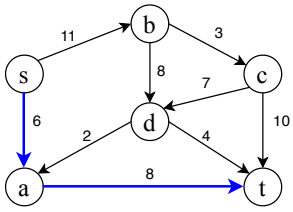
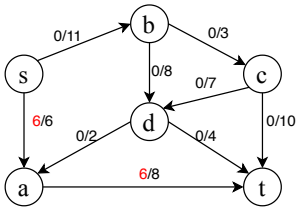
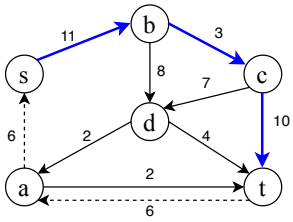
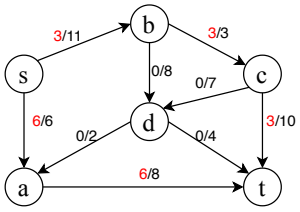
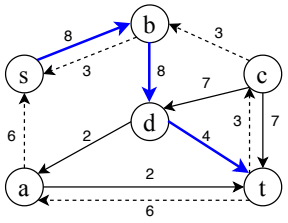
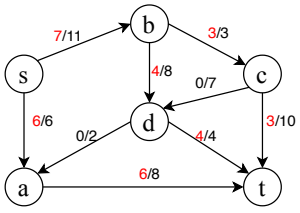
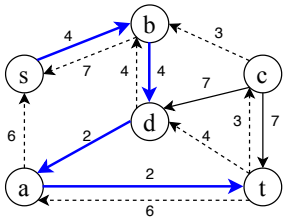
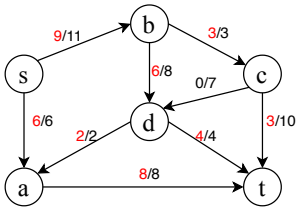
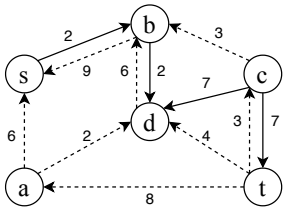

   (a) Run the Ford-Fulkson algorithm on this instance. For each iteration, illustrate the flow $f$ (i.e., $f(e)$ for every $e \in E$), the augmenting path $p$, the bottleneck capacity $x(p)$, and the value of the flow at the end of the iteration.

   (b) Let $f^*$ to be the maximum-flow found in (a). Give the $s$-$t$ cut $(S^*, T^*)$, where $S^* := \{v \in V \mid s \text{ can reach } v \text{ in } G_{f^*}\}$, and $T^* := V \backslash S^*$. Show the cut-edges and the capacity of this $s$-$t$ cut.

   (c) Let $f^*$ to be the maximum-flow found in (a). Give the $s$-$t$ cut $(S', T')$, where $T' := \{v \in V \mid v \text{ can reach } t \text{ in } G_{f^*}\}$, and $S' := V \backslash T'$. Show the cut-edges and the capacity of this $s$-$t$ cut.

   Solution:

   (a) See the process below. The augmenting path $p$ in the current iteration is marked as bold blue arrows, and the dashed arrows indicate backward edges in the residual graph $G_f$.

| | Flow $f$ | Residual graph $G_f$ & Augmenting path $p$ | $x(p)$ | $\lvert f\rvert$ |
|---|---|---|---|---|
| Iteration 1 |  |  | 6 | 0 |
| Iteration 2 |  |  | 3 | 6 |
| Iteration 3 |  |  | 4 | 9 |
| Iteration 4 |  |  | 2 | 13 |
| Final |  |  | / | $\lvert f^*\rvert = 15$ |

(b) $S^* = \{s, b, d\}$, $T^* = \{t, c, a\}$. Cut edges: $\{(s, a), (d, a), (d, t), (b, c)\}$, capacity of cut: 15.

(c) $S' = \{s, a, b, d\}$, $T' = \{t, c\}$. Cut edges: $\{(b, c), (d, t), (a, t)\}$, capacity of cut: 15.

2. (12 pts.) You are given a directed graph $G = (V, E)$ with positive integral capacity $c(e)$ on $e \in E$, a source $s \in V$, and a sink $t \in V$. You are also given an integral maximum $s$-$t$ flow $f$ of $G$. Now we pick one edge $e^* \in E$ and reduce its capacity by 1. Design an algorithm to find a maximum flow in the resulting network in $O(|E| + |V|)$ time.

Solution. Assume $e^* = (u, v)$. We denote by $G'$ the network after reducing $c(e^*)$ by 1; denote by

$f'$ one maximum-flow of $G'$. Consider $f(e^*)$. There are two cases: $f(e^*) < c(e^*)$ and $f(e^*) = c(e^*)$. For the first case, reducing $c(e^*)$ by 1 does not invalidate $f$. So for this case $f' = f$ remains the maximum flow of $G'$. Now consider the second case. For this case the value of the maximum-flow for $G'$ either does not change, i.e., $|f'| = |f|$, or reduce by 1, i.e., $|f'| = |f| - 1$. We now design an algorithm to decide which one is the case and construct the corresponding $f'$. We build the residual graph $G_f$ with respect to $f$ for $G$. In $G_f$, we use DFS/BFS to determine if there exists a path $p$ from $u$ to $v$.

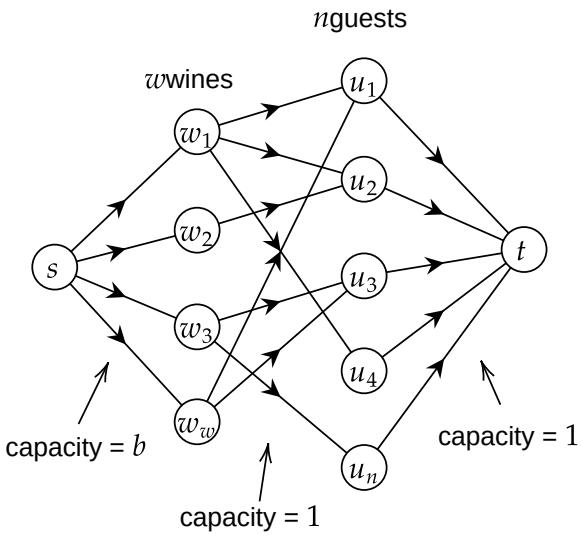
If such path $p$ exists, then we can construct $f'$ such that $|f'| = |f|$. Specifically, since $f(e^*) = c(e^*)$, in $G_f$ there exists a backward edge $(v, u)$. The path $p$ combined with $(v, u)$ forms a cycle $C$ in $G_f$. We then modify $f$ in $G$ following this cycle $C$: if an edge $e$ is a forward edge, we increase $f(e)$ by 1; otherwise we decrease $f(e)$ by 1. (This modification follows the same procedue as we augment a path in the Ford-Fulkson algorithm.) The resulting flow, denoted as $f'$, must be a flow of $G$, since the modification follows edges in the residual graph so all capacity constraints must be satisfied. Also, because these modifications follow the cycle, every vertex must remain balanced (same proof with the Ford-Fulkson algorithm). Last, we must have $|f'| = |f|$, again because the modification follows a cycle so that no vertex (including $s$ or $t$) gain or loss any flow. Now, note that the edge $e^*$ is not saturated wrt $f'$, since $f'(e^*) = f(e^*) - 1$ as in $G_f$ the edge $(v, u)$ is a backward edge. So, $f'$ is also a valid/maximum flow for $G'$.

If such path $p$ does not exist, then it must be $|f'| = |f| - 1$. To construct $f'$, in $G_f$ we find a path $p_1$ from $u$ to $s$, and find a path $p_2$ from $t$ to $v$. Both paths $p_1$ and $p_2$ must exist (because $f(e^*) = c(e^*) > 0$). Also $p_1$ and $p_2$ must not overlap as otherwise there will be a path from $u$ to $v$ in $G_f$. Let $p'$ be the path from $t$ to $s$ by concatenate $p_2$, edge $(v, u)$, and $p_1$. We then augment each edge $e$ in $p'$: if $e$ is a forward edge, increase $f(e)$ by 1, and otherwise decrease $f(e)$ by 1. The resulting flow is denoted as $f'$. Certainly $f'$ is a flow of $G'$ and its value satisfies $|f'| = |f| - 1$.

3. (12 pts.) There are $n$ guests, $w$ different kinds of wines in one party. Each guest has a preference list that specifies the wines they like. A guest is well-served if they are served a wine from their preference list. Each kind of wine can be served to at most $b$ guests. Describe a polynomial-time algorithm to compute the maximum number of guests that can be well-served. Hint: reduce this problem to a network flow problem.

   Solution. This problem can be reduced to a network flow problem. The network $G$ has one vertex $w_i$ for the $i$th wine, $1 \le i \le w$, one vertex $u_k$ for the $k$th guest, $1 \le k \le n$, and source $s$ and sink $t$. The edges of the network are $(s, w_i)$ with capacity $b$, $(w_i, u_k)$ with capacity 1 if $i$th wine is in the preference list of $k$th guest, $(u_k, t)$ with capacity 1. See example below.

   We compute the maximum flow $f^*$ of the above network $G = (V, E)$. Since all capacities of $G$ are integers, $|f^*|$ must be an integer and $f^*(e)$ must be an integer too for any $e \in E$. We now prove that $|f^*|$ is the maximum number of the guests that can be well served. This is rather straightforward. On one side, suppose that $k$ guests can be well served. Then we can construct a flow $f$ of $G$ with value $|f| = k$, simply setting $f(w_i, u_k) = 1$ if $k$th guest is well-served and served with $i$th wine, and setting $f(s, w_i)$ and $f(u_k, t)$ to the proper value to balance $w_i$ and $u_k$. This proves that the maximum number of guests can be well served is a lower bound of $|f^*|$. On the other side, we can find $|f^*|$ guests that can be well served following $f^*$: those guests with $f^*(w_i, u_k) = 1$ can be well served and the corresponding wines can be fetched following $f^*(w_i, u_k) = 1$. This proves that the maximum number of guests can be well served is an upper bound of $|f^*|$. Combined, $|f^*|$ equals to the maximum number of the guests that can be well served. The network has $(w + n + 2)$ vertices and $O(wn)$ edges. Thus, this algorithm, whose running time is dominated by calling a max-flow solver, runs in polynomial-time.


$n$guests
$w$wines
$u_1$
$w_1$
$u_2$
$w_2$
$s$
$t$
$u_3$
$w_3$
$u_4$
$w_w$
$u_n$
capacity = $b$
capacity = 1
capacity = 1