**CS218 Design and analysis of algorithms** **Jan-Apr 2025**

# Quiz 2

*Total Marks: 20* *Time: 55 minutes*

## Instructions.

- Please write your answers clearly and concisely.

**Question 1.** Let $J_1, J_2, \ldots, J_n$ be job positions and let $C_1, C_2, \ldots, C_m$ be candidates. Each candidate has a strict preference order over job positions and each job position has a strict preference order over candidates. Assume $m = n$ for now. For a given perfect matching $M$ between jobs and candidates, we defined a blocking pair as a pair $(J_i, C_k)$ such that

- $J_i$ is matched with $C_{k'}$ in $M$ and $J_i$ prefers $C_k$ over $C_{k'}$
- and $C_k$ is matched with $J_{i'}$ in $M$ and $C_k$ prefers $J_i$ over $J_{i'}$.

A perfect matching $M$ is called stable if there is no blocking pair with respect to $M$.

Now suppose number of jobs and candidates are not necessarily same. Also, a candidate (or job position) may not be interested in all job positions (or candidates). Each candidate has a strict preference order among the job positions she/he is interested in. Similarly, each job position has a strict preference order over candidates it is interested in. Naturally, each job/candidate prefers to be matched to something in their list rather than being unmatched. But, it may not be possible to match every job/candidate. We want to define stable matching in this more general setting, again via blocking pairs.

**(a) [3 marks].** In addition to the above defined blocking pair, how should we define other possible blocking pairs, with respect to any given matching $M$?

Hint: There are three kinds of blocking pairs other than the above. Pairs $(J_i, C_k)$ where $J_i$ is unmatched, or $C_k$ is unmatched, or both are unmatched.

- $(J_i, C_k)$ is a blocking pair if $J_i$ is unmatched and has $C_k$ in its list and $C_k$ is matched $J_{i'}$ and prefers $J_i$ over $J_{i'}$.
- $(J_i, C_k)$ is a blocking pair if $C_k$ is unmatched and has $J_i$ in its list and $J_i$ is matched $C_{k'}$ and prefers $C_k$ over $C_{k'}$.
- $(J_i, C_k)$ is a blocking pair if both $J_i$ and $C_k$ are unmatched and have each other in their list.

Marks have been deducted if it is not mentioned that "has $J_i$ in its list" etc.

**(b) [2 marks].** Consider the following example for jobs $J_1, J_2, J_3$ and candidates $C_1, C_2, C_3$. Below are their preferences. If a job (or candidate) is not present in a candidate's (or job's) list, then they cannot be matched with each other.

- $J_1 : C_3 > C_2$
- $J_2 : C_3 > C_2 > C_1$
- $J_3 : C_2 > C_3 > C_1$
- $C_1 : J_2 > J_3$
- $C_2 : J_2 > J_1 > J_3$
- $C_3 : J_3 > J_1 > J_2$

Consider a matching where $J_2$ is matched to $C_3$, and $J_3$ is matched to $C_2$. Job $J_1$ and candidate $C_1$ remain unmatched. Is this a stable matching? Explain why or why not.

No.

It is not stable because $(J_1, C_2)$ is a blocking pair. Because $J_1$ is unmatched and has $C_2$ in its list and $C_2$ prefers $J_1$ to its current match $J_3$.

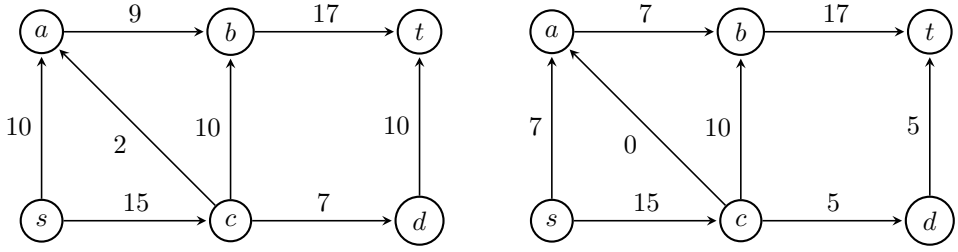
Alternatively, it is not stable because $(J_1, C_3)$ is a blocking pair. Because $J_1$ is unmatched and has $C_3$ in its list and $C_3$ prefers $J_1$ to its current match $J_2$.

1 mark for answer, 1 mark for explanation. Showing any one of the two blocking pairs is sufficient.

**(c) [3 marks].** Give the job-optimal stable matching for the above preference lists. Recall that in the job-optimal stable matching, every job gets matched to the best possible candidate it could get in any stable matching. (We had seen an algorithm for this in the class.)

Stable matching is $(J_2, C_2), (J_3, C_3)$, the remaining $J_1$ and $C_1$ are unmatched.

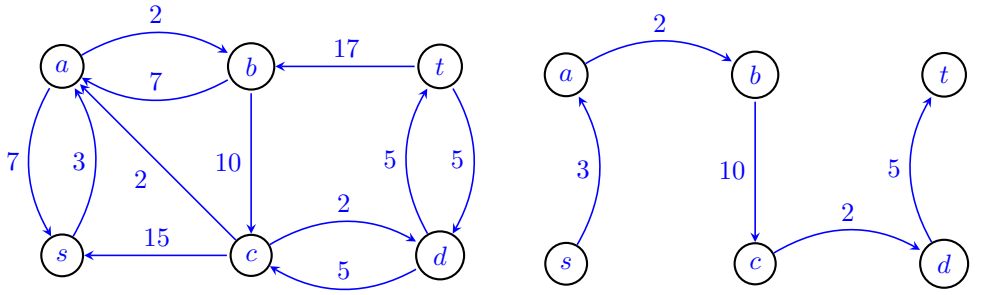
**Question 2.** Consider the flow network in Figure (1a) with a source $s$ and a sink $t$. The edge capacities are shown on the edges. Figure (2b) shows an $s$-$t$ flow in the same network with flow values indicated on the edges.




(a) A flow network with capacities on the edges (b) Current flow values shown on the edges.

Figure 1: Network Flow

**(a) [2 marks].** Construct the residual graph with respect to the flow shown in Figure (2b) and capacities shown in Figure (1a). You do not need to show any (forward/backward) edges with zero residual capacity.




(a) Residual capacities shown on the edges (b) A path from $s$ to $t$ in the residual graph.

Figure 2: Network Flow

**(b) [1+1 marks].** Find a path from $s$ to $t$ in the residual graph (using edges with nonzero residual capacity)? What is the maximum possible additional flow you can push along this path?

$s \to a \to b \to c \to d \to t$.

The minimum residual capacity along this path (bottleneck) is 2. Hence, 2 units is the maximum possible flow we can push along this path.

**(c) [2 marks].** After pushing this additional flow, show the updated flow value on each edge of the given graph.

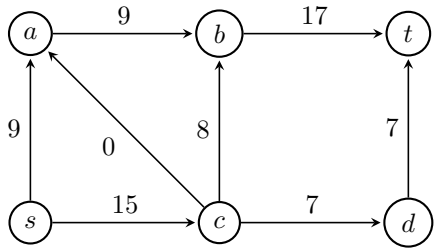



Figure 3: Network Flow

**(d) [1+1 marks].** Is it the maximum possible flow from $s$ to $t$? Give a short argument for your answer.

Yes, it is the maximum possible flow. The total flow going out of $s$ is 24, while there is an $s$-$t$ cut $\{s, a\}$ whose outgoing capacity is 24. Since flow is upper bounded by the capacity of any $s$-$t$ cut, 24 is the maximum.

**Question 3 [4 marks].** We are given a flow network with integer capacities (directed graph, single source $s$ with no incoming edges, single sink $t$ with no outgoing edges). Suppose the maximum possible outgoing flow from source is $x$ units. We want to increase it to $x + 1$ units by increasing capacities of certain edges (not allowed to add new edges). We want to minimize the number of edges whose capacities we have to increase. Give a polynomial time algorithm for this. You can directly use subroutines for problems discussed in this or previous courses.

Hint: example in the previous question may be helpful.

- Compute maximum flow $f$ using the algorithm seen in the class.
- Compute the residual graph $G_f$ with respect to max flow $f$ (zero capacity edges deleted).
- Since it is maximum flow, there is no way to increase the flow. That means there is no path in the residual graph from $s$ to $t$.
- Note that we need to increase capacity of only those edges in the given network, which are saturated (flow = capacity). That is, edges whose residual capacity is zero. When we increase an edge's capacity by 1 unit, its forward residual capacity will also increase by 1 unit. While backward capacity remains same.
- In the residual graph, add all the forward edges with zero residual capacity. Mark these edges as red.
- Now, in this modified graph, we find a path from $s$ to $t$ with minimum number of red edges.
- This can be done via Dijkstra's algorithm by giving red edges weight 1, the other edges weight 0, finding minimum weight path.
- Note that the desired path may involve backward edges. The algorithm is correct only if you also consider backward edges. Clearly, zero capacity backward edges should not be considered.
- Note that BFS/DFS cannot find a path with minimum number of red edges.