We conclude our discussion of network flows with an application to bipartite matching. We need the following definitions:

A graph \( G(V, E) \) is a bipartite graph if \( V \) can be partitioned into two sets \( A \) and \( B \), such that \( A \cup B = V \), and for all \( e = (a, b) \in E \), \( a \in A, b \in B \).

The pair \( (A, B) \) is called a bi-partition. We’ll use \( G(A \cup B, E) \) to refer to a bipartite graph with the bi-partition \( (A, B) \).

A matching in a graph is a set of edges \( M \subseteq E \) such that for every pair of edges \( e_1, e_2 \in M \), \( e_1 \) and \( e_2 \) do not share a common end point.

A matching is maximal if it cannot be extended to a larger matching, and maximum if it has the most edges out of any matching \( M' \) of \( G \).

Example: The graph below is a bipartite graph with bi-partition \( A = \{a,b,c\} \) and \( B = \{x,y,z\} \). The matching \( M = \{(a, x), (b, y)\} \) is maximal, whereas \( M' = \{(a, z), (b, y), (c, x)\} \) is maximum.

So the problem we are trying to solve is the following:

**The Bipartite Matching Problem:**

**Input:** A bipartite graph \( G(A \cup B, E) \).

**Output:** A maximum matching \( M \) of \( G \).

But why this problem and how is it related to network flow? This is just to illustrate how Ford-Fulkerson can be applied in different ways. To solve this problem, we will give a reduction from the bipartite matching problem to the maximum flow problem. That is, we will (1) somehow change our bipartite matching problem into a max-flow problem, compute the max flow \( f \) and (2) use this solution (the flow \( f \)) to extract a solution to our original problem, namely finding a maximum matching.

In more precise terms: Given any bipartite graph \( G(A \cup B, E) \), we will show how to construct a flow network \( (G', s, t, c) \) such that \( \text{max}(f) = |M| \), where \( M \) is a maximum matching for \( G(A \cup B, E) \). We will then use this max flow \( f \) on \( G' \) to reconstruct the corresponding maximum matching \( M \) on \( G \). This happens in two steps, which we will present in two separate algorithms:
Algorithm 1 ConstructNetwork

Input: A bipartite graph $G(A \cup B, E)$

Output: A flow network $(G', s, t, c)$ such that $val(f)$, the max flow $f$ of $G'$, equals the size of a maximum matching on $G$.

1: Construct a flow network $(G', V', E')$ as follows:
2: $V' = A \cup B \cup \{s, t\}$  \(\triangleright\) The vertices of $G'$ are the same as the vertices of $G$, plus a source and a sink
3: For every $a \in A$, add the edge $(s, a)$ to $E'$
4: For every $b \in B$, add the edge $(b, t)$ to $E'$
5: For every $(a, b) \in E$, add $(a, b)$ to $E'$
6: Set the capacity $c(e') = 1$ for every edge $e' \in E'$

Algorithm 2 ExtractMatching

Input: A flow network $(G', s, t, c)$ and a max flow $f$ on $G'$

Output: A maximum matching $M$ on $G$

1: for every edge $(a, b) \in G'$ with flow $f(a, b) = 0$ do
2: Remove $(a, b)$ from $G$.
3: end for
4: Return the set of edges with $f(a, b) = 1$.

What’s the complexity of these two algorithms? Well, ConstructNetwork takes $O(m + n)$; one iteration through the vertices and edges of $G$ suffices, and ExtractMatching takes $O(m)$ time to select the edges with nonzero flow. The final algorithm looks like this now:

Algorithm 3 Bipartite Matching

Input: A bipartite graph $G(A \cup B, E)$

Output: A maximum matching $M$ on $G$

1: $(G', s, t, c) \leftarrow \text{ConstructNetwork}(G)$
2: $f \leftarrow \text{FordFulkerson}(G', s, t, c)$
3: $M \leftarrow \text{ExtractMatching}(G', f)$
4: Return $M$

and takes $O(m^2n)$ time. Why?

Let’s prove that all of this actually works. To show that the reduction works, we’ll prove the following theorem:

Theorem: Let $G(A \cup B, E)$ be a bipartite graph. Let $(G', s, t, c)$ be a flow networks constructed by ConstructNetwork on $G$:

1. The size of the maximum matching $M$ of $G$ equals the value of the maximum flow $f$ on $G'$.
2. ExtractMatching returns a maximum matching when given the max flow $f$ on $G'$.

Proof. 1. Let $f$ be the maximum flow on $G'$ and let $a$ be any element of $A$. Notice that if $f(s, a) = 1$ then there must exist a vertex $b$ such that $f(a, b) = 1$ (by conservation of flow). By construction of $G'$, we know there exists an edge $(b, t)$ with capacity 1, therefore $f(b, t) = 1$. So this augmenting path $s - a - b - t$ matched vertex $a$ to vertex $b$. Since $c(s, a) = 1$, we know there doesn’t exist a vertex $b'$ with $f(a, b') = 1$ otherwise
we violate the conservation of flow property. So $a$ is matched to $b$ only. And using the same argument on $b$, we conclude that $b$ is matched to $a$ only. Therefore, the set $M = \{(a,b) \in G'|a \in A, b \in B, f(a, b) = 1\}$ must be a matching in $G$.

2. Now suppose $M$, the matching returned by ExtractMatching is not maximum on $G$, and let $M'$ be a maximum matching for $G$. Therefore $|M'| > |M|$.

Now construct a flow $f'$ as follows: For every $(a, b) \in M'$, we set $f'(a, b) = 1$. By the previous argument, $f'$ is a valid flow in $G'$, and $\text{val}(f') = |M'| > |M| = \text{val}(f)$. This contradicts the maximality of $f$ on $G'$!