1. Definitions:
   (a) Matching on $G$: a set of edges without common vertices
   (b) Maximum matching on $G$: a matching of maximum cardinality over all possible
       matchings on $G$ (different from a maximal matching!)
   (c) Free vertex: $v$ s.t. no edge on matching $M$ is incident to $v$
   (d) Alternating path: path $P$ in $G$ such that every other edge in $P$ is in the
       matching $M$
   (e) Augmenting path: alternating $P$ with start and end vertices free
2. Berge’s Lemma: a matching $M$ in $G$ is maximum iff no augmenting paths exist.
   (a) If $G$ has an augmenting path $P$, consider $M' = P \oplus M$. Observe $M'$ is a
       matching with $|M'| = |M| + 1$.
   (b) Suppose $M$ is not maximum, i.e. there exists matching $M'$ with $|M'| > |M|$. 
       Consider $D = M \oplus M'$. $D$ must consist of even cycles and paths whose edges
       alternate between $M$ and $M'$. Since $M'$ is larger, $D$ contains some path that
       starts and ends with an edge from $M'$, i.e. an augmenting path.
3. This gives us the general idea for finding a maximum matching on $G$:
   (a) Set $M \leftarrow \phi$
   (b) Set $P \leftarrow \text{find_augmenting_path}(G, M)$
   (c) If $P$ non-empty, set $M \leftarrow M \oplus P$ and go to (b).
   (d) Else, return $M$
4. Easy: find_augmenting_path on bipartite graphs
   (a) Create auxiliary structure $M$-alternating forest – initialize $F$ as singleton trees
       with roots = exposed vertices in $M$; label these vertices ‘red’
   (b) For each red vertex $u$, iterate through all edges $(u,v) \notin F$. Two cases:
       i. $v$ is not in $F$, i.e. $v$ must be in $M$. Suppose $(v,v')$ is matched edge in $M$.
          Add $(u,v)$ and $(v,v')$ to $F$, label $v$ ‘blue’ and $v'$ red.
       ii. $v$ is in $F$. Then $v$ must be red and in a different tree from $u$ (why?). Root
           of $u \rightarrow \ldots \rightarrow u \rightarrow v \rightarrow \ldots \rightarrow$ root of $v$ is an augmenting path, return it.
(c) If no red vertices are connected to each other, no augmenting paths exist, return $\phi$.

5. Runtime of `find_augmenting_paths` is $O(m)$, so runtime to find maximum matching is $O(mn)$.

6. Hard: `find_augmenting_path` on non-bipartite graphs (i.e. the blossom algorithm)
   (a) Same as before, except possible to have $(u, v) \not\in F$, with $u$ and $v$ red and in same component of $F$
   (b) Call odd cycle containing $u$ and $v$ the blossom (i.e. a cycle of length $2k + 1$ with $k$ edges in $M$, call path from root of tree to blossom the stem
   (c) Blossom lemma: suppose $M$ matching in $G$ and $B$ a blossom. Contract $B$ to one vertex, call reduced graph $G'$. Then there exists augmenting path on $G$ iff there exists augmenting path on $G'$.
   (d) Strategy if encountering blossom:
      i. Contract $B$ to a vertex, recurse to find augmenting path $P'$ on $G'$
      ii. Lift $P'$ to $P$ on $G$, return $P$

7. Runtime analysis:
   (a) $O(n)$ possible augmenting paths in total
   (b) $O(n)$ blossoms to shrink along each augmenting path
   (c) $O(m)$ work to identify a blossom or augmenting path.
   (d) $O(m)$ work to expand a blossom
   (e) $O(mn^2)$ runtime in total