Consider a bipartite graph $G$ on two sets of vertices, the left hand side and the right hand side. We define a **matching** of $G$ to be a collection of disjoint edges (no vertex appears in two edges) between the two sets. A matching is **perfect** if every vertex on the left hand side is matched with some other vertex on the right hand side.

**Example 0.1.** We have a list of tasks that need to be performed, and of people who can do some of the tasks. Each task needs only one person to do it, and each person can only do one task. However, not everyone can do every task. We can think of this as a graph where the tasks are on the left hand side, people are on the right hand side, and we connect a task to a person if that person can perform the task. A matching corresponds to an assignment of some of the tasks. The matching is perfect if every task gets done.

In general, if we're given a graph we want to know if a matching exists, and how to find it.

This can be viewed as a special case of the placing rooks on chessboards. Suppose we view the vertices on one side as rows and the vertices on the other side as columns, then shade each square that doesn't have an edge between its row and column. We're then reduced to the following problem: Given an $m \times n$ board, can we place $m$ non-attacking rooks on its shaded squares? In theory we could solve this problem by computing the rook polynomial of the board using the methods we learned earlier, then seeing if the $x^m$ coefficient is zero or not. In practice, this method is awful because computing rook polynomials takes too long. So we look for a different method.

We first note that there is one natural condition which in and of itself is already enough to guarantee that a matching is not possible. Consider the graph below:

```
A——B——C——D——E
|     |     |     |     |
|     |     |     |     |
|     |     |     |     |
|     |     |     |     |

The top three vertices on the left hand side between them only have two neighbors on the right hand side, so there’s no hope of matching all three of them at once. With this in mind, we make the following definition

**Definition 0.2.** Given a subset of $A$ of the vertices on the left hand side of our graph, we let $R(A)$ be the vertices on the right hand side that are adjacent to at least one vertex in $A$.

What we mentioned above is the following fact:
Fact 0.3. If there is at least one subset $A$ such that $R(A)$ has fewer elements than $A$, then there cannot possibly be a perfect matching.

As it turns out, this necessary condition also turns out to be sufficient, a result originally due to Philip Hall in 1935:

Theorem 0.4. If for every subset $A$ we have that $R(A)$ is at least as large as $A$, then there is a perfect matching.

While this result is nice in theory, it at first glance seems very unsatisfying for two reasons. First, suppose we’re trying to match 100 people to 100 jobs. Checking the condition directly would require checking $2^{100}$ subsets, a near-impossible task. Furthermore, even if we did check the condition, we would rather actually find the matching than just know that one exists. We’ll answer both of these objections by our proof of the theorem, which will consist of an algorithm that either finds the matching or finds a subset which has too few neighbors and tells us that there is not a matching.

Let us imagine the edges of the graph as being colored either blue (unused) or red (used). Our goal will be to inductively build up a collection of red edges which form a perfect matching. The key concept we will use here is that of an alternating path, by which we mean a path in the graph that begins by following a blue edge, then a red edge, then a blue edge, and so on, finishing by following a blue edge.

Example 0.5. Any path of length 1 consisting of a blue edge is alternating, as is the path to the right.

Informally, these alternating paths represent the following process. We take a job $a$ that hasn’t been matched yet, and give it to a person $z$ (the blue edge from $a$ to $z$). If $z$ hasn’t been matched yet, great. If person $z$ was already assigned to task $b$ (a red edge from $b$ to $z$), then we need to find $b$ a worker (a blue edge from $b$ to $y$). If $y$ hasn’t been matched yet, great, if not we continue onwards, and so on. We can stop once we match one of our jobs to a person who did not previously have a match. In terms of our colored edges, we have the following

Fact 0.6. If there is an alternating path beginning at an unmatched job and ending at an unmatched person, then we can swap the edge of every color on the path. Doing so gives us a situation where one more person is matched than before.

This suggests the skeleton of an algorithm: If everyone is matched, stop. If this is not the case, find an alternating path and swap the colors to increase the number of matched people. However, we still have two questions: How can we be sure that there is such an alternating path in the first place? Even if there is one, how can we find it?

So what we need is the following “path finding” algorithm:
- Initialize $A$ to be a set consisting of one unmatched vertex, and $B$ to be empty.
- Find a vertex $y$ in $R(A)$ that is not in $B$. If $y$ is unmatched, then stop. If $y$ is matched, then add $y$ to $B$ and add the match of $y$ to $A$.

See the graph below for an example:

```
initialize: $A = \{a1, a2\}$ B empty + y.
blue edge $a1 \rightarrow y$, $y$ matched + to e.
$A = \{a1, e\}$, $B = \{y\}$.
blue edge $e \rightarrow z$, $z$ matched + to g.
$A = \{a1, e, g\}$, $B = \{y, z\}$.
blue edge $e \rightarrow u$, $u$ matched + to b.
$A = \{a1, e, g, a2\}$, $B = \{y, z, b\}$.
blue edge $b \rightarrow x$, $x$ matched + to d.
$A = \{a1, b, e, g\}$, $B = \{y, z, b, d\}$.
blue edge $g \rightarrow w$, $w$ unmatched. Stop.
```

The key observations about this algorithm are:

- $B$ always contains all of the vertices previously matched with $A$ (connected to $A$ by a blue edge), since we only add a vertex to $A$ after we added its match to $B$. In particular, the neighbor of $y$ must not have already been in $A$ before we added it.
- $A$ is always larger than $B$. If this algorithm gets stuck, it happens because all of the vertices in $R(A)$ are already in $B$. But this means that $R(A)$ is smaller than $A$, so there isn’t a matching.
- Every time we add a new vertex $y$ to $B$, it’s connected to a vertex that was previously in $A$ by a blue edge, and a vertex that we are newly adding to $A$ by a red edge. The only exception is when the algorithm stops.

Once the algorithm stops, we just follow the path backwards from our unmatched vertex on the right hand side to the unmatched vertex we started with. This gives us an alternating path. To summarize, our algorithm for constructing a matching is as follows:

**Step 0:** Start with all edges in the graph colored blue.
**Step 1:** Using the path-finding algorithm, find an alternating path starting at an unused vertex on the left hand side and finishing at an unused vertex on the right hand side. If no such path exists, no matching is possible.

**Step 2:** Swap the colors of every edge on the alternating path.

**Step 3:** Repeat steps 1 and 2 until all vertices on the left are matched.

We finish with a couple applications of Hall’s Theorem:

**Example 0.7.** Deal out all 52 cards of a deck into 4 rows of 13 cards each. Then you can always choose a card from each column in such a way that the chosen cards all have different ranks.

To see this, we construct a bipartite graph there the left hand side consists of the 13 columns, and the right hand side consists of the thirteen ranks of cards. We connect a column to a rank if a card of that rank occurs somewhere in the column.

If we consider a set of $k$ columns, than it contains $4k$ cards. Those cards must be of at least $k$ different ranks, since there are only 4 cards of each rank. In other words, for any set $A$ of $k$ columns, $R(A)$ must be at least as large as $A$. By Hall’s theorem, there is a matching, which gives what we want. More generally, we have

**Theorem 0.8.** Let $k$ be at least 1. Any $k$-regular bipartite graph has a perfect matching.

The proof of this is similar to the proof above. If we consider a set of $c$ vertices on the left hand side, then there are $k \times c$ edges leaving the set. But only $k$ edges go into any vertex, so those edges must go to at least $c$ different places. In other words, if $A$ has $c$ vertices, than $R(A)$ has at least $c$ vertices. But this implies that there is a matching.