Problem 1

Given an edge $e = (a_1, a_2)$, where $a_1 \in V_1$, $a_2 \in V_2$, here is an algorithm to generate a perfect matching of G that contains e. So, we will keep track of S_1, S_2, and M, where S_1 contains the vertices in the matching so far from V_1, S_2 contains the vertices in the matching so far from V_2, and M contains the edges in the matching so far. So to start, M will just contain e, and S_1 and S_2 will contain the respective vertices. Then we will iterate through the vertices in $V_1 - a_1$. For each vertex v, if it has an edge to a vertex not already in S_2, then we can just add that edge and its respective vertices to M, S_1, and S_2. If all of vs edges are to vertices in S_2, then we can do the following. Consider the set of vertices $S = S_1 + v - a_1$. We have that S must be connected to two vertices not in $S_2 - a_2$; otherwise, we would have that $|S| \geq |N(S)|$. So then, we can just add one of those two vertices to S_2 (if one of them is a_2, take the other one so we don't disconnect e), add v to S_1, and add/switch the edges around in M accordingly. That is, we basically switch a vertex in S to match with something else (one of those two vertices not in $S_2 - a_2$) so that v can match with a vertex in $S_2 - a_2$. Note that in this process, we never lose the edge e from our matching, since one of the two vertices we found was a_2, we picked the other one.

So that is our process. We have that we can continue this process until we add the final edge to our matching which makes it perfect. It is clear that we can do this because our algorithm always uses that fact that $|A| < |N(A)|$ only for subsets of V_1 that are not empty or not V_1 itself. Thus, given an arbitrary e, we have shown that we can find a perfect matching of G that contains e. So we have proved the desired statement.

Alternatively, we can use Hall's Theorem to prove it. So given an arbitrary $e = (a_1, a_2)$, consider the graph $G' = (V_1 - a_1 \cup V_2 - a_2, E - S)$, where S is the set of all edges adjacent to a_1 and a_2. Let $|N'(X)|$ denote the cardinality of the neighborhood of X in G', and $|N(X)|$ denote the cardinality of the neighborhood of X in G. Then we have that $|A'| < |N(A')|$ (property of the original graph) and that $|N'(A')| \geq |N(A')| - 1$ (since the only neighboring node we remove in G' is a_2) which tells us that $|A'| \leq |N'(A')|$. This means that G' has a perfect matching by Hall's Theorem. Then we have that there is a perfect matching for G because we can just add e to the perfect matching of G' to make it. And since for any edge e we can do this, we have that for every edge $e \in E$, there exists a perfect matching of G that contains e.

Problem 2

We will format this problem as a graph problem. Let one set of vertices V_1 be those vertices that represent each distinct type of card, and let one set of vertices V_2 be those vertices that represent each column in the $n \times m$ matrix. So we have that $|V_1| = |V_2| = m$. Let G be the connected undirected bipartite graph $G = (V_1 \cup V_2, E)$, where E are the edges connecting V_1 and V_2. We have that for each vertex $v \in V_1$, there are n edges going from it to vertices in V_2. This is because each distinct type of card appears n times, so it can appear in at most n columns. Then we have that for each vertex $v \in V_2$, there are n edges going into it from vertices in V_1. This is because each column has n rows, so each column can have n cards. Then we have that by the variant of Hall's Theorem shown in lecture 12, we have a perfect matching (a matching of size $|V_1|$), since for every subset A of V_1 we have that $|A| \leq |N(A)|$. This fact is clear given that each vertex in V_1 has n outgoing edges and each vertex in V_2 has n incoming edges. Therefore, we can conclude that there is always a matching for G that matches every vertex in V_1 with a vertex in V_2. Notice that this the same as saying that every column can be matched with a distinct type of card. But due to our formulation, this means that there exists a set of m cards such that each is of a different type and each is in a different column. To elaborate, our graph represents all the ways that the m distinct types of cards can fall into the m columns. And since we always have a perfect matching in our graph (which means each column can be matched with a distinct type), the desired statement is true. Thus we have proved the desired statement.

Problem 3

Here is the strategy:

1. Biscuit finds a maximum matching M.
2. Biscuit starts by choosing a vertex v_1 not in M.

3. Biscuit, when it is his turn, always chooses an edge that is in M.
4. Biscuit wins.

For this problem, we will define an "unmatched vertex" as a vertex that is not adjacent to any edge in M.

Before proving our strategy works we will first prove a lemma. The lemma is that if we have a maximum matching M, then there does not exist an path in M that starts and ends on unmatched vertices. So, consider an arbitrary matching M, and assume to the contrary that there is such a path p that starts and ends on unmatched vertices. Then we can increase the cardinality of our matching by first taking all the edges in p out of our matching, then adding back in alternating edges along p, starting with the first edge of p. We have that this keeps M a matching because we alternate in p, and the start and end vertices of p were unmatched to begin with, so making those matched is fine. And we have that this increases the cardinality of M because before in p, the maximum number of edges we could match was by alternating edges, not including the starting and ending edge (because the starting and ending vertices were unmatched). So now that we alternate along the whole path, we can clearly match more, since before we were alternating along a path shorter by two edges.

Now we will prove that our strategy works using our lemma. So, we have that Biscuit starts by choosing a vertex v_1 not adjacent to any edge in M. Then we have that by our lemma, Adam will travel to a vertex v_2 adjacent to an edge in M. Then Biscuit will choose an edge that is in M. Then Adam will choose an edge not in M, then Biscuit will always choose an edge that is in M. Now, given this strategy, assume to the contrary that Adam wins. This means that Adam ended on a vertex that is not adjacent to any edge in M - otherwise, Biscuit would have picked it. But this contradicts our lemma, since Biscuit started on an unmatched vertex. So we have that Biscuit must win.

Problem 4

We will prove this to be true. So, assume to the contrary that if the size of the maximum flow is not a multiple of five, then not every maximum flow of the network has a non-zero flow through e. That is, there exists some maximum flow that has zero flow through e. Let us call this flow f. We have that since the size of the maximum flow is not a multiple of five, the size of the min cut is not a multiple of five (by the max flow min cut theorem). This clearly means that the min cut goes through e, since all the other edges have capacities that are multiples of five. Then we have that e must be saturated in f, since the cut goes through e (we proved this in lecture 13, slide 14). But this is a contradiction, since we said that f had zero flow through e. Thus we have proved our statement.

Problem 5

Here is our algorithm.

1. Give all edges in the graph $G = (V, E)$ a capacity of ∞.
2. Make all vertices, except for s and t, into edges with capacity 1. So if we have a vertex v, it will become an edge $e = (a, b)$, where the capacity of e is 1. Call this modified graph $G' = (V', E')$, where $E' = E + X$, where X are the edges spawned by the vertices.
3. Find the edges in the min cut of G' and return the vertices that spawned those edges. To find the min cut, we can first find the capacity of the min cut using Ford-Fulkerson (covered in class). We can then use BFS to find a cut whose capacity is equivalent to the capacity of the min cut (this was also covered in class).

Now we will prove that this algorithm is correct. We have that the min cut disconnects the graph by definition (and thus that the set of vertices we pick disconnects the graph). We also have that it disconnects the graph by minimizing the capacity of the edges it cuts. This means that the edges we cut will have to be in X, since the edges in E have infinite capacity. Then we have that since we assigned all the edges in X to have a capacity of 1, and the edges in X actually represent vertices in G, finding the min cut in G' is the same as finding the minimum set of disconnecting vertices in G (given how we constructed G'). Thus our proof is complete.