Automata Theory (CSE4058-01)
Homework # 05 - Solution

4.2 If we have a partial k-coloring c of G, such that $c(v) \in \{1, \ldots, k\}$ for some vertices v and $c(v)$ is undefined for others, our goal is to define a new G' that is k-colorable if and only if c can be completed to a proper k-coloring. Then we can add one vertex at a time to c, by trying its k possible colors.

Here's one way to do this. Add k new vertices, labeled $1, \ldots, k$, forming a complete graph K. Then construct G' from $G \cup K$ as follows. For each vertex v which is colored in c, merge v with vertex $c(v)$ in K. (Alternately, draw an edge between v and the $k-1$ vertices in K other than $c(v)$.) This forces all the vertices in G with a given $c(v)$ to have the same color (since they are now the same vertex), while forcing those with different $c(v)$ to have different colors (since there is an edge between them in K). If G' is k-colorable, then G has a k-coloring which agrees with c.

Another approach is to start with the graph G, choose two vertices which are not adjacent, and try to merge them together. If G is still k-colorable after we do this, it means there is a coloring where these two vertices are the same color. If not, these vertices have to be different colors (perhaps because of decisions we made earlier) so we should un-merge them. Keep merging pairs of vertices until only k vertices are left. Then the vertices of G which became the ith of these k vertices can be colored i.

4.12 This is a classic example of the pigeonhole principle. There are $2^{10} = 1024$ different subsets, but their totals all lie in the range 0 to 1000. Since $1024 > 1001$, it is impossible for each of the subsets to have a different total. Therefore, there must be two sets A and B with the same total. Removing $A \cap B$ from both of them makes them disjoint, while preserving the fact that they have the same total.

4.16 We will show that the size of the smallest vertex cover equals the size of the maximum matching. This reduces these problems to MAX BIPARTITE MATCHING, and shows that they are in P.

First, let S be a vertex cover, and let M be a matching. At least one endpoint of each edge in M must be included in S, so $|S| \geq |M|$. Thus the minimum vertex cover is at least as large as the maximum matching, or $\min|S| \geq \max|M|$.

However, showing that $\min|S| = \max|M|$ takes more work. There are several ways to do this, but we will use the reduction from MAX BIPARTITE MATCHING to MAX FLOW and the duality between MAX FLOW and MIN CUT. Add vertices s and t to the left and right sides of the graph as in Figure 3.23 on page 75. As in the proof of Hall's theorem in Problem 3.47, there is a minimum cut consisting of edges connecting s to some set X of vertices on the left, and edges connecting t to some set Y of vertices on the right. There are no edges from $\overline{X}$ to $\overline{Y}$, since otherwise there would be a path from s to t. Therefore, $S = X \cup Y$ is a vertex cover of the bipartite graph.

This shows that there is a vertex cover of size equal to the minimum cut, which is also the size of the maximum flow, and therefore of the maximum matching. We already showed that any vertex cover has to be at least this large, so we're done.

4.17 Suppose a graph G has a vertex cover S of size k. If we remove an edge, then S is still a vertex cover. If we contract an edge (u, v), then at least one of u or v must have been in S. If we include the new, combined vertex in the vertex cover, then it covers all the edges that u or v had before. In either case we get a vertex cover S' of the new graph G', where $|S'| \leq |S|$.

For the explicit algorithm, choose any edge (u, v). Branch into two subproblems. In one, include u in S and remove u and its edges from the graph. In the other, do the same with v. In each case, ask whether the remaining graph has a vertex cover of size $k-1$. Working recursively produces a binary tree of depth k. At each leaf, we ask whether a graph has a vertex cover of size 0, which it does if and only if it has no edges. Each step requires poly(n) bookkeeping to remove a vertex from the graph, so the total running time is 2^kpoly(n).