CS 3510 Honors Algorithms
Solutions : Homework 3

Problem 1. [**Testing for Bipartiteness**]

Assume that the graph is connected. If the graph is not connected, then we can use the same test for each of the different connected components. We will check if the vertices can be labeled 0 or 1 such that all the edges are connecting a 0-vertex to a 1-vertex. Use DFS as follows : Start from a root vertex v, labeling it with 0. As we encounter an edge going from vertex i to j, then label$(j) = 1 - $ label(i). If we ever try to label a vertex different to what we have already labeled it as, we reject saying that the graph is not bipartite. We accept when we finish traversing all the edges and do not find a contradiction. If all the edges can be labeled where one vertex is 0 and the other one is 1, this means that the labels 0 and 1 correspond to two sets such that every edge is connecting two vertices of different sets.

```
v.label := 0 for vertex v;
w.label :=-1 for all vertices w which is not v;
Number_visited_edges = 0;
Edges = 0 for all edges;

Bipartite(G, v, Edges, Number_visited_edges) {
   if (Number_visited_edges = |E|) exit and return BIPARTITE
   for all neighbors w of v do {
       if (w.Label=v.label) exit and return NOT-BIPARTITE
       w.label = 1 - v.label
       if Edges(v,w)=0  {
           Edges(v,w)=1
           Number_visited_edges = Number_visited_edges + 1
      }
       Bipartite(G, w, Edges, Number_visited_edges)
   }
}
```

For the correctness of the algorithm, notice that if we try to label a vertex different from what it already is labeled as, then we reject the graph saying it is not bipartite. Else we succesfully label all the vertices, and accept it, after we are done going through all the vertices.

For the running time, notice that we run through each edge exactly once, and we take a constant amount of time for each edge. So the running time is $O(|E|)$.

Problem 2. [**Minimum Spanning Trees**]

(a) Prim's algorithm will work. Let us see why. Consider a graph G', which is the same as G, but with the difference that the end points v_1 and v_2 of e have been contracted to form a single vertex v. v has edges to all the vertices which had edges to either v_1 or v_2 or both, with the weight of the edges being the same. If both v_1 and v_2 were connected to the same vertex, replace it with an edge to v having weight the minimum of the two weights. A tree T' in G' would give rise to a tree in G. Also, the minimum weighted tree in G which contains e would correspond to an MST in G' and vice versa. Running the Prim's algorithm on G' would give us the required tree.

    We could achieve this in practice by starting Prim's algorithm with both v_1 and v_2 marked and edge e has already been picked.

    Kruskal's algorithm will also work due to the same reasoning. We could first pick e and then sort the remaining edges and run through them, adding and forming the required tree.

(b) Assume that there are two different minimum spanning trees T and T' for G. Let e' be the lightest edge in one of the trees but not in the other. Without loss of generality, let e' be the lightest edge in T' which is not there in T. By the exchange property (at the end of the Berkeley notes) there exists an edge $e \in T - T'$ such that $(T - \{e\}) \cup \{e'\}$ is also a spanning tree. But since e' was the lightest edge in $(T - T') \cup (T' - T)$, and all the edge weights are different, e must have a larger weight than e' and thus $(T - \{e\}) \cup \{e'\}$ must have a smaller weight than T, which contradicts the fact that T is a minimum spanning tree.

Problem 3. [**More MST**]

First note that we can obtain any MST by running Kruskal's algorithm by using a certain ordering of edges when it comes to deciding the order of edges with the same weight. In particular, we can use an ordering that puts edges in T first whenever there is a tie. Running Kruskal's algorithm on G we can get T.

Assume that Kruskal's algorithm was run through the edges of H ordered in the same way that they appeared in the above run of G. We claim now the algorithm will add all the edges in $H \cap T$ into the MST which it outputs.

Let us assume on the contrary that there exists some edge $(u, v) \in H \cap T$ that is not added. In the ordering of the edges for Kruskal's algorithm, let P' be the set of all edges in H processed before (u, v) and let P be the set of all edges in G processed before (u, v). Since (u, v) is not added into the MST for H, u and v must already be connected in the forest that we are building. So there is some set of edges in P' that could connect u and v. So the same edges must also be in P. These edges can be used in T to connect u and v, unless there exists lighter weighted edges in P which connect u and v. Then adding (u, v) to T would create a cycle, which is a contradiction to the fact that (u, v) was a valid edge in the MST T created by Kruskal's algorithm on G.

Problem 4. [**Mutual Acquaintances**]
If there is a person a who knows at least 3 other people, then if none of these people do not know each other, we are done because we have a set of 3 people none of who know each other. Otherwise, if there is a pair of people who know each other, these two people along with a, form a triplet of people who all know each other, we are done.

Now let us take the case when every person knows atmost 2 other people. We can represent them in a graph, with an edge between a and b if and only if they know each other. Notice that every vertex has degree atmost 2.

Let us take the case when degree is exactly 2. Then with 6 points and degree 2, we can get only a 6 cycle (we are done here by taking alternate people who don't know each other) or two triangles (we have two sets of 3 people who know each other).

When any of the vertices has a degree 1, it means that that person does not know four other persons. Either all four of these people know each other (we are done), or there is a pair among these who don't know each other (we are done in this case as well).