Solutions for Homework Assignment #5

Answer to Question 1.

Conceptually, the main idea is to find some spanning tree of the graph and remove one of the leaves in the spanning tree. Then, the rest of the graph is still connected (using the rest of the spanning tree).

More formally, we give a constructive proof that every connected graph contains at least one vertex whose removal does not disconnect the graph. Consider the following algorithm:

> Run DFS on the input graph G, modified so that when DFS-VISIT is called on a node whose neighbours are all non-white, it immediately returns that node (returning NIL otherwise).

This modification involves performing a constant amount of additional work for each invocation of DFS-VISIT so the resulting algorithm still runs in linear time (as a function of the size of the input graph). Moreover, the vertex returned is guaranteed *not* to disconnect G when it is removed. This is because when this vertex is reached, all its neighbours have *already* been discovered through other paths. Finally, the algorithm is guaranteed to eventually reach such a vertex; the opposite would mean that G is infinite as every vertex would always have at least one non-discovered neighbour.

Answer to Question 2.

a. Suppose, for contradiction, that G is bipartite but has a cycle of odd length, say $u_0, u_1, \ldots, u_{2k}, u_0$, for some $k \geq 1$. Let V_0 and V_1 be the two sets into which the set of nodes of V is partitioned so that every edge joins a node in V_0 to a node in V_1. Without loss of generality, assume that $u_0 \in V_0$. Then all nodes on the cycle with even subscript are in V_0 and all nodes with odd subscript are in V_1. (To see this, suppose the contrary, and consider the node u_i with smallest index that violates this property. This node cannot be u_0, since we assumed that $u_0 \in V_0$. But then, u_{i-1} and u_i are both in V_0 or both in V_1, contrary to the definition of a bipartite graph.) Hence u_{2k} and u_0 are both in V_0. But these nodes are adjacent, so we get a contradiction to the assumption that G is bipartite.

b. We want to prove that if G has no simple cycle of odd length, then G is bipartite. It suffices to restrict our attention to connected graphs G: G has no simple cycle of odd length if and only if every connected component of G has no simple cycle of odd length; and G is bipartite if and only if every connected component of G is bipartite. So, let G be a connected graph in which no simple cycle has odd length. We will prove that G is bipartite.

Let s be any node of G. Perform a BFS on G starting at s, and let T be the resulting BFS tree. For each node u of G, let $d[u]$ be the distance of u from s (i.e., the length of a shortest path from s to u), as computed by the BFS of G. Let V_0 be the set of all nodes u of G such that $d[u]$ is even, and V_1 be the set of all nodes u of G such that $d[u]$ is odd. Because G is connected, every node of G is in V_0 or V_1.

To prove that G is bipartite, it suffices to prove that every edge of G connects a node in V_0 to a node in V_1. Suppose, for contradiction, that G has an edge (u, v) that connects two nodes in V_0 or two nodes in V_1. Therefore $d[u]$ and $d[v]$ have the same parity (both are even or both are odd).

Let w be the *lowest common ancestor (lca)* of u and v in the BFS tree T.[1] Let P_u be the path on T from s to u and let P'_u be the subpath of P_u from w to u; similarly, let P_v be the path on T from s to v and let P'_v be the subpath of P_v from w to v. Denote the length of a path P by $|P|$.

Since u and w are both in V_0 or both in V_1, $|P_u|$ and $|P_v|$ have the same parity. (This is because, as we proved in class, the path from s to each node x in the BFS tree is a shortest path from s to x; therefore, $|P_u| = d[u]$ and $|P_v| = d[v]$, and $d[u], d[v]$ have the same parity.) Let P_w be the path on T from s to w. Note that P_u consists of P_w followed by P'_u; and similarly P_v consists of P_w followed by P'_v. Therefore $|P'_u| = |P_u| - |P_w|$ and $|P'_v| = |P_v| - |P_w|$. Since $|P_u|$ and $|P_v|$ have the same parity, so do $|P'_u|$ and $|P'_v|$.

[1] The lca of two nodes u and v in a tree T is the node w such that (a) w is an ancestor of both u and v; and (b) any node w' that is an ancestor of both u and v is an ancestor of w. It is easy to show that such a node always exists: Consider the path from the root to u and the path from the root to v. The first node after which these two paths diverge is their lca w. Note that, because T is a tree, two paths that diverge cannot meet again.

The edges of P'_u and P'_v, together with the edge (u, v) form a simple cycle in G, whose length is $|P'_u| + |P'_v| + 1$. Since $|P'_u|$ and $|P'_v|$ have the same parity, $|P'_u| + |P'_v|$ is even, and so $|P'_u| + |P'_v| + 1$ is odd. This contradicts that G has no simple cycle of odd length — thus, there is no edge (u, v) that connects two nodes in V_0 or two nodes in V_1.

c. The algorithm works as follows:

(1) Perform a BFS starting at any undiscovered node x (i.e., node x such that $d[x] = \infty$).

(2) For each edge (u, v) explored by the BFS, if at the time when (u, v) is explored from node u, $d[v] \neq \infty$ (i.e., v has already been discovered) and $d[u], d[v]$ have the same parity, then the algorithm returns "not bipartite".

(3) When the current BFS ends, if there is still some unvisited node, then repeat steps (1) and (2). Otherwise, let V_0 be the set of nodes u such that $d[u]$ is even and V_1 be the set of nodes u such that $d[u]$ is odd, and return (V_0, V_1).

The running time of this algorithm is dominated by the time to perform BFS until all nodes are discovered. This takes $O(n + m)$ time, where n is the number of nodes and m is the number of edges of the graph.

For the correctness of the algorithm we reason as follows. There are two cases.

1. The algorithm returns "not bipartite". We must prove that, in this case, G is not bipartite. Since the algorithm returns "not bipartite", there is an edge (u, v) so that, when the BFS algorithm explored (u, v), it found that $d[u]$ and $d[v]$ have the same parity. Let w be the lowest common ancestor of u and v in the BFS tree. Arguing as in part (b), the edges on the BFS tree path from w to u and w to v, together with the edge (u, v) form a simple cycle of odd length in G. By part (a), G is not bipartite.

2. The algorithm returns a pair of sets of nodes (V_0, V_1). We must prove that, in this case, every edge (u, v) of G connects a node in V_0 to a node in V_1. Indeed, $d[u]$ and $d[v]$ have opposite parity: If (u, v) is a BFS tree edge, this is because $d[u]$ and $d[v]$ differ by one. If (u, v) is not a BFS tree edge, this is by the algorithm (otherwise the algorithm would have returned "not bipartite"). Since $d[u]$ and $d[v]$ have opposite parity, one of u, v is in V_0 and the other is in V_1.

Answer to Question 3.

a. (i) We construct a complete graph G by creating a vertex for each bike pump station and an undirected edge between every two vertices. The weight on edge $e = (u, v)$ is the Euclidean distance between u and v; that is, $w(e) = \sqrt{(x_v - x_u)^2 + (y_v - y_u)^2)}$, where (x_u, y_u) and (x_v, y_v) are the coordinates of u and v, respectively.

(ii) Our task restated as a graph problem: find a path in G from vertex s to vertex t where the weight of the largest edge is minimal (over all paths from s to t), i.e., finding the path in G with minimum maximum edge weight among all paths from s to t

b. The $s \to t$ path where the largest edge weight is minimized is the $s \to t$ path in some MST of G. (Since the MST is a tree, it has only one path from s to t.)

Proof. Let P be an $s \to t$ path in some MST T of G. We claim that P is a path with the minimum maximum edge weight among all the paths from s to t. Assume, for a contradiction, that this is not the path from s to t with the minimum maximum edge weight, that is, there is some *other* path P' that connects s and t and has a strictly smaller maximum edge weight. Remove from T the edge e_M that has the maximum edge weight in P. (This edge exists because P is a path of this MST.) This disconnects our MST T; i.e. we're considering a cut of our MST: $C = (S, V - S)$ where $s \in S$ and $t \in V - S$. There must be an edge e in P' that crosses this cut (since this path forms a path from s to t). Furthermore, by our assumption on P', the maximum weight of the edges in P' is strictly smaller than the weight of e_M. Thus, since e is an edge in P', $w(e) < w(e_M)$. We can then replace e_M with e in the MST, and obtain a spanning tree of total weight less than our MST. This is a contradiction.

c. We find a Minimum Spanning Tree of G by running either Prim's or Kruskal's algorithm. While we execute this algorithm, we construct the adjacency list of the MST that this algorithm finds. We then execute the Breadth-First Search algorithm on this MST (using its adjacency list computed in the previous step) to a find the path from s to t on this MST.

d. Constructing our graph G takes $\Theta(|V| + |E|) = \Theta(n + n^2) = \Theta(n^2)$. Constructing an MST from G with either Prim's or Kruskal's algorithm (using the adjacency list representation of G) takes $O(|E| \log |V|) = O(n^2 \log n)$. The MST of G has $|V|$ vertices and $|V| - 1$ edges (do you see why?). So running the BFS algorithm on our MST takes $\Theta(|V| + (|V| - 1)) = \Theta(n)$. Thus, the worst-case running time of our algorithm is dominated by the worst-case running time of the MST algorithm, and so it is $O(n^2 \log n)$.