CSE 202 Homework 2
Combinatorial Search and Optimization:
Graph Algorithms Sample Solutions

1. **Broadcast Times for a Tree**

   Problem Let T be a rooted binary tree whose edges are given positive real weights, representing *message delay times*. For a node $x \in T$, define the *broadcast time* for x to be the maximum over $y \in T$ of the weighted distance in T between x and y. (In other words, the broadcast time is the time before a message originatin at x is received by every other processor y.) Give an efficient algorithm to compute all the broadcast times for nodes in T simultaneously.

   Solution

   This solution basically performs two *level traversals* of the tree. One upward from the leaves keeping track of the largest broadcast up times and one downward adding on the broadcast down part of a path.

   Associate with each edge, (u, v), a variable $d[(u, v)]$ that will hold the broadcast time first from the farthest node in the direction of v (the child), then from the farthest in the direction of u. Starting at the leaves and working in levels toward the root, set

   $$d[(u, v)] = max_{j \in C(v)} \left(d[(v, j)] + w[(u, v)]\right)$$

   All $d[(u, v)]$ will then hold the longest broadcast time from u to a vertex in the subtree rooted at v. Then from the root down to the leaves set the broadcast time of vertex i to be the farthest time either down one of its children or up through it's parent,

   $$b[i] = max_{j \in N(i)} \left(d[(i, j)]\right)$$

   Also, set for i's children, $j \in C(i)$, broadcast times on the edges now reflecting upward broadcasts,

   $$d[(i, j)] = max_{k \in N(i), k \neq j} \left(d[(i, k)]\right) + w[(i, j)]$$

   This algorithm makes two passes over the $n - 1$ edges in the tree and performs work linear in the degree of each vertex, d. The instances in the problem are stated to be binary trees, so $d \in O(1)$ and the total time is $O(n)$. (This can be shown to be $O(n)$ for a tree with any degrees by amortizing the work over the edges, which are set twice and examined three times each).

   We can prove the correctness by looking at the content of the variables $d[(u, v)]$. In the first pass (from the leaves to the root), we claim that each $d[(u, v)]$ is set to the longest broadcast time from u to some vertex in the subtree rooted at v (possibly v itself). Clearly this is true for the base case when v has no children by setting $d[(u, v)]$ to $w[(u, v)]$. Recursively, the maximum downward broadcast from u through v, $d[(u, v)]$, is the maximum broadcast from v down any of the children of v added to the delay $w[(u, v)]$ as the recursion states. This completes the first pass of the algorithm. Then, when we traverse down the tree

and are at vertex u, we maintain that $d[(parent(u), u)]$ holds the broadcast time from u to the farthest node in the subtree $T - subtree(u)$, and the $d[(u, v)]$ are still the down-broadcast times. This is trivially true for the root, since it has no parent and $T - subtree(root) = \emptyset$. Recursively, the farthest node from v in $T - subtree(u)$ must come from the parent side of u or from one of its children, which the recursion provides. Finally, the farthest node from i is either connected to i through its parent or its children. These farthest broadcast times are exactly those contained in the d variables for the neighbors of i and the broadcast times are thus correct.

2. Prove that a graph is bartite if and only if it has no odd cycles.

   $\Rightarrow$ Assume that G has an odd cycle $C = (c_1, c_2, \ldots, c_n, c_1 = c_{n+1})$ (n is odd). Since G is bipartite, we can place verticies with edges between them into two disjoint sets, $left$ and $right$. Arbitrarily select c_1 on the $left$. Since consecutive verticies in the path c_i, c_{i+1} are connected, c_{i+1} must be on the side c_i is not. Therefore, since c_1 is in $left$, all even verticies must be in $right$. But $c_1 = c_{n+1}$ is an even vertex in the chain and must be in $right$; c_1 is already in $left$ and since $right$ and $left$ are disjoint we have a contradiction.

   $\Leftarrow$ Constructively, for each connected component begin with an arbitrary vertex i. Place i in $left$. Place each neighbor of $j \in left$ in $right$ and then each neighbor of $j \in right$ in $left$ repeatedly until all verticies are in $left$ or $right$. We claim there is no edge between $i, j \in left$ or $i, j \in right$. Assume this is not true and WLOG there is an edge $i, j \in left$ and j was added after i. Necessarily, j has a set of neighbors, J_1, in $right$ and J_1 has a set of neighbors, J_2, in $left$. If $i \in J_2$ then this is an odd cycle contradicting the assumption. Else, this repeats, $J_{2k} \subseteq left, J_{2k+1} \subseteq right$. Since this is a connected component (and j was clearly not added to $left$ by our process as a neighbor of i), for some k, we must have $i \in J_{2k}$ and thus we have a cycle $(i, j, j_1, \ldots, j_{2k})$, where $j_i \in J_i$ and $j_{2k} = i$. This cycle is of odd length contradicting the assumption.

3. **Maximum matching in a forest**

   Lemma In a forest, F, if a leaf node, u, exists and has a neighbor v, then a maximum matching on F exists that contains (u, v).

   Proof Assume a different maximum matching N is found on the forest not containing the edge (u, v). That matching must necessarily contain an edge (v, x) where $x \neq u$ (otherwise, it is not a maximum matching since (u, v) can be added to it). Further no other edge (v, y), $y \neq x$ is in the matching since that would violate the matching condition. Therefore, we can remove (v, x) and replace it with (u, v) without changing the cardinality of the matching and without violating the matching condition.

   Algorithm proof (Inductively on the number of edges) If F has no edges then all matchings are of zero cardinality and are maximum. Then applying the above theorem, take any leaf u with neighbor v and a maximum matching,

M, on F exists with (u, v). Further, this matching is the same size as any maximum matching, $M' + (u, v)$, where M' is on $F - \{(v, x) : x \in N(v)\}$ which has fewer edges than F. Therefore, the induction holds and the algorithm returns a maximum matching for a forest of any size.