CHAPTER 13

EXERCISE 13.1. Observe that a spanning tree \( T \) which minimizes the objective function \( [\Sigma_{(i,j) \in T} (c_{ij})^2]^{1/2} \) must also be optimal when the objective function is \( [\Sigma_{(i,j) \in T} (c_{ij})^2] \). To optimize the later objective, identify a spanning tree in the network \( G \) with the cost of each arc \( (i, j) \) as \( (c_{ij})^2 \).

EXERCISE 13.3. When Kruskal's algorithm is applied to the network shown in Figure 13.15(a), then the order in which the arcs be included in the spanning tree is (7, 8), (5, 6), (1, 4), (1, 3), (5, 8), (9, 8), (3, 6) and (2, 4). The optimal spanning tree is illustrated in Figure S13.3(a). In the case of Figure 13.15(b), the order in which arcs will be included in the spanning tree is (6, 9), (2, 3), (2, 6), (5, 6), (5, 7), (5, 8), (1, 2) and (4, 5). The final spanning tree is illustrated in Figure S13.3(b).

EXERCISE 13.5. When Sollin's algorithm is applied to the network illustrated in Figure 13.15(a) in the text, then the sequence of forests obtained is illustrated in Figures S13.5(a) through (c). In the case of Figure 13.15(b) of the text, the sequence of forests obtained is illustrated in Figures S13.5(d) through (f).
EXERCISE 13.7. The approach described in Application 13.3 cannot be generalized to directed networks, because a spanning tree cannot contain a directed path from i to j as well as a directed path from j to i; for a network containing both of these paths would contain a directed cycle.

EXERCISE 13.9. The proof techniques for the maximum spanning tree optimality conditions are exactly analogous to those of the minimum spanning tree optimality conditions discussed in the text.

EXERCISE 13.11. Suppose that the maximum weight subgraph $G^0$ of the graph $G$ is not a spanning tree. Then, the subgraph must be a forest with at least two components. Let $S$ be the node set corresponding to any one of the components. However, since $G$ is a connected graph, it must have at least one arc $(i, j)$ in common with $[S, S']$. Addition of arc $(i, j)$ to $G^0$ produces a forest but strictly increases the weight of $G^0$ because the weight of arc $(i, j)$ is strictly positive. This contradicts the fact that $G^0$ is a maximum weight subgraph of $G$. Hence, $G^0$ must be a spanning tree.

EXERCISE 13.13. A minimum spanning tree of $G$ need not necessarily be a tree of shortest paths. In the example illustrated in Figure S13.3(a), the length of the path from node s to node r in the minimum spanning tree (shown in Figure S13.3(b)) is not a shortest path.

EXERCISE 13.15. Consider any arc $(i, j)$ which lies in $T'$ but does not lie in $T''$. Removal of the arc $(i, j)$ from $T'$ creates a node partition $[S, S']$ which must contain at least one arc $(k, l)$ of $T''$. Removal of arc $(i, j)$ and addition of arc $(k, l)$ to $T'$ creates a tree $T^1$ which is adjacent to $T'$, but has one more arc in common with $T''$. By repeating this procedure, we can find a sequence of spanning trees $T' = T^1, T^2, ..., T^K = T''$, with $T^i$ adjacent to $T^{i+1}$ for every $i = 1$ to $k-1$.

EXERCISE 13.17. Given any particular root node $s$, the value of $b[s, j]$ (for all nodes $j$) can be found using any search algorithm starting from node $s$. When the algorithm examines the arc $(i, j)$ such that $b[s, i]$ has already been computed, then it sets $b[s, j] = \min\{ b[s, i], c_{ij} \}$. Since the search algorithm requires $O(n)$ time on a tree, and search algorithm have to be applied $O(n)$ times (by varying the root node), the algorithm requires $O(n^2)$ time. Observe that an initialization of $b[i, i] = \infty$ (for all $i$) is required by the algorithm.

EXERCISE 13.19. It is easy to see that at the termination of the algorithm, the graph $G'$ obtained is a spanning tree because the algorithm examines every arc in the network and keeps deleting them as long as the network remains
connected. Hence, at the termination of the algorithm no cycle could remain in the network, because the algorithm will delete the first arc which it examines in that cycle. As the network is connected and contains no cycles, it must be a spanning tree. Now consider any cycle \( W \). Observe that since the algorithm examines arcs in the nonincreasing order of their lengths, the first arc examined in the cycle \( W \) would be the largest cost arc. This implies that the fundamental cycle introduced by the nontree arcs satisfy the optimality conditions and, consequently, the resulting tree will be a minimum spanning tree.

**Exercise 13.21.** If the data structure of Dial’s implementation (as described in Section 4.6) is used to implement Prim’s algorithm, then \( k \) th bucket would store all the nodes whose distance labels equal \( k \). We need a total of \( C \) buckets because each \( d(i) \leq C \). To identify a node with the smallest distance label, we scan the buckets 0, 1, 2, ..., until we find a nonempty bucket. We point out that in every iteration, we examine the buckets 0, 1, 2, ... contrary to Dial’s implementation where we examine at the place left-off earlier in the previous iteration, because the minimum distance labels are nonnecessarily nondecreasing. In this implementation, identifying minimum distance labels requires a total of \( O(nC) \) time and updating the buckets requires a total of \( O(m) \) time. Hence, the running time of this implementation is \( O(m + nC) \), which is the same as for Dial’s implementation for the shortest path problem.

**Exercise 13.23.** The initial value of the potential function \( \Phi = \sum_{(i,j) \in T} f(i,j) \) is nonnegative and the final value of \( \Phi \) is at most \( nm \). Hence, when we replace a tree arc \((i, j)\) by a nontree arc \((k, l)\), then \( c_{kl} < c_{ij} \), which implies that \( f(k, l) \geq f(i, j) + 1 \). Hence each arc exchange increase \( \Phi \) by at least one and the algorithm performs \( O(nm) \) iterations.

**Exercise 13.25.** (a) When \((i, j) \in T^*\), then the maximum value of the cost of the arc \((i, j)\) is equal to the minimum value of the cost of any nontree arc in the cut formed upon removing \((i, j)\) from \( T^* \). If \((k, l)\) be such a nontree arc, then the cost interval of the arc \((i, j)\) is \([-\infty, c_{kl}]\). Clearly, this method can be implemented in \( O(m) \) time. When \((i, j) \notin T^*\), then the minimum value of the cost of the arc \((i, j)\) is equal to the maximum value of the cost of any tree arc in the cycle \( W \) formed by the addition of arc \((i, j)\) to \( T^* \). If \((k, l)\) be such a maximum cost tree arc in \( W \), then the cost interval of arc \((i, j)\) is \([c_{kl}, \infty]\). Clearly, this method can be implemented in \( O(n) \) time.

(b) If we apply the method described in part (a), then we determine the cost intervals of all arcs in \( O(nm) \) time. We can, however, do it in \( O(n^2) \) time using the following method. For any pair \([i, j]\) of nodes, let \( \alpha[i, j] \) denote the maximum cost arc in the tree path joining node \( i \) and node \( j \). By a slight modification of the method suggested in the solution of Exercise 13.17, the value of \( \alpha[i, j] \) may be obtained for every pair of nodes in \( O(n^2) \) time. Then, the cost interval of any non-tree arc \((i, j)\) is simply \([\alpha[i, j], \infty]\).

We now explain how we can determine the cost intervals of all the tree arcs of a minimum spanning tree \( T^* \) in a total of \( O(m \log n) \) time. There is a much simpler \( O(n^2 + m \log n) \) algorithm, but we provide the \( O(m \log n) \) algorithm for completeness. Suppose that we order the nontree arcs in a nondecreasing order of the costs. Let these arcs be denoted by \( a_1, a_2, a_3, a_4, \ldots, a_{m-n+1} \). For each tree arc \((i, j)\), we want to find the minimum cost nontree arc in the cut formed upon removing \((i, j)\) from \( T^* \). (In the case of tie, choose the arc \( a_k \) with minimum index \( k \).) It is easy to see that arc \( a_k \) is the minimum cost nontree arc for the tree arc \((i, j)\) if the following conditions are satisfied: (i) \((i, j)\) is on the path in \( T^* \) joining the endpoints of \( a_k \); and (ii) \((i, j)\) is not on the path in \( T^* \) joining the endpoints of \( a_{k'} \) for \( k' < k \). Using this idea, we have the following algorithm to determine the cost intervals of the tree arcs:
unmark all tree arcs;

for \( k = 1 \) to \((m-n+1)\) do

begin

select the nontree arc \( a_k \);

mark all the unmarked tree arcs joining the endpoints of \( a_k \) and set its \( \alpha \) value equal to \( a_k \);

end;

Sorting of the nontree arcs takes \( O(m \log n) \) time. We now explain how the above described algorithm can be implemented in \( O(m + n \log n) \) time. Notice that at any point in the algorithm, the tree arcs can be partitioned into two subsets: marked or unmarked. The collection of all nodes and the subset of marked arcs define a forest. We maintain the nodes in each component of the forest as a linked list as in the description of the improved version of Kruskal’s algorithm. Thus, each node \( i \) will have a pointer \( \text{First}(i) \) that is the first node in its component. Let us call these nodes the root nodes. Also, each root node \( j \) will have a pointer \( \text{Top}(j) \) which will be the least depth node in the component containing the root node \( j \). To maintain \( \text{Top}(j) \) takes \( O(1) \) additional steps whenever two components are merged since \( \text{Top}(\cdot) \) only changes for root nodes.

We claim that each node in the component is a descendent of \( \text{Top}(j) \). This is clearly true for components with one node. In general, we will merge the components containing \( i \) and \( j \) for some arc \((i,j)\) in the tree. Suppose that \( j = \text{pred}(i) \). Then \( i \) must be the topmost element in its component since its topmost element is an ancestor of \( i \) and \( j \) every proper ancestor of \( i \) is also an ancestor of \( j \). Thus after the merging, all nodes will be a descendent of the \( \text{Top}(\text{First}(j)) \).

Finally, let \( \text{Depth}(j) \) denote the depth of node \( j \) in the original tree. The algorithm for finding all unmarked arcs from \( i \) to \( j \) is as follows.

In the above algorithm, suppose that we want to whenever we select the arc \((i,j) = a_k\).

begin

\[ u := \text{Top}(\text{First}(i)); \]

\[ v := \text{Top}(\text{First}(j)); \]

while \( u \neq v \) do

begin

...
if Depth(u) ≤ Depth(v) then

begin

mark arc (u,pred(u));
merge the components containing u and pred(u);
let u := Top(First(pred(u)));

end

else if Depth(u) > Depth(v) then

begin

mark arc (v,pred(v));
merge the components containing v and pred(v);
let v := Top(First(pred(v)));

end

end

The algorithm essentially scans from node i up the tree and from node j up the tree until a common ancestor is discovered. However, the algorithm skips over all marked arcs. If the current node is u, then the next arc scanned is (u',pred(u')) where u' = Top(First(u)) which is the topmost node in the component containing u.

The running time of the algorithm is O(m + n log n) except for the sorting. The time to scan arcs is O(m) since each arc is scanned but once. The time to maintain components is O(n log n).

EXERCISE 13.27. Arc Deletions: If the arc being deleted is a non-tree arc, then the minimum spanning tree $T^*$ does not change. On the other hand, if the arc (i, j) being deleted is a tree arc, then we obtain in O(m) time a minimum cost arc in the cut $[S, \overline{S}]$ formed by removing arc (i, j) from $T^*$. The new minimum spanning tree is simply the one formed by replacing arc (i, j) in $T^*$ by the arc (k, l).

Arc Additions: Let $P$ be the unique path from node i to node j in $T^*$ and arc (k, l) be a maximum cost arc in $P$. If $c_{ij} \geq c_{kl}$, then $T$ remains optimal; otherwise replacing arc (k, l) by the arc (i, j) yields an optimal tree of the modified problem. As |P| ≤ n, this method takes O(n) time.

EXERCISE 13.29. (a) A minimum spanning tree for the network shown in Figure S13.29(a) is given in Figure S13.29(b).
(b) Let \( \alpha_p = \min \{ \alpha_i : i \in N \} \). Then the minimum spanning tree consists of the arcs \((p, 1), (p, 2), \ldots, (p, n)\) excluding the arc \((p, p)\). It is easy to verify that every nontree arc satisfies its path optimality condition.

\[ c_{ij}^{*} \leq c_{k,l}^{*} \quad \text{for each tree arc } (i, j) \in P[k, l], \quad (i) \]

where \( P[k, l] \) is the unique tree path from node \( k \) to node \( l \) in \( T^k \). Observe that when \( k \) is sufficiently large (in fact, any number \( k > \max \{ c_{ij}^{0} : (i, j) \in A \} \) will do), then (i) is satisfied if and only if \( c_{ij}^{*} \leq c_{k,l}^{*} \). Hence, \( T^k \) is a minimum spanning tree with \( c_{ij}^{*} \) as arc lengths. Also observe that when \( k \) is sufficiently small (in fact, any number \( k \leq -\max \{ c_{ij}^{0} : (i, j) \in A \} \) will do), then (i) is satisfied if and only if \( c_{ij}^{*} \geq c_{k,l}^{*} \), implying that \( T^* \) is a maximum spanning tree with \( c_{ij}^{*} \) as arc lengths.

EXERCISE 13.33. Let the arcs be indexed in the nondecreasing order of their costs; let the distinct arc costs be denoted as \( c_1 < c_2 < \ldots < c_p \), where \( p \leq m \). The bottleneck spanning tree problem is to identify the smallest index \( k \) so that the subgraph comprising of arcs with less than or equal to \( c_k \) contains a spanning tree. Notice that using any search algorithm, we can determine in \( O(m) \) time whether or not a subgraph contains a spanning tree. Clearly, we can use a binary search technique to determine the smallest index \( k \) for which the corresponding subgraph contains a spanning tree. Thus this approach solves the bottleneck spanning tree problem in \( O(m \log p) = O(m \log n) \) time.

EXERCISE 13.35. Consider the minimum spanning tree \( T^k \) for sufficiently large value of \( k \). The path optimality conditions imply that for every nontree arc \((k, l)\) in \( T^k \),

\[ c_{ij}^{0} + k c_{ij}^{*} \leq c_{k,l}^{0} + kc_{k,l}^{*} \quad \text{for each tree arc } (i, j) \in P[k, l], \]

where \( P[k, l] \) is the unique tree path from node \( k \) to node \( l \). Observe that when \( k \) is sufficiently large (in fact, any number \( k > \max \{ c_{ij}^{0} : (i, j) \in A \} \) will do), then (i) is satisfied if and only if \( c_{ij}^{*} \leq c_{k,l}^{*} \). Hence, \( T^k \) is a minimum spanning tree with \( c_{ij}^{*} \) as arc lengths. Also observe that when \( k \) is sufficiently small (in fact, any number \( k \leq -\max \{ c_{ij}^{0} : (i, j) \in A \} \) will do), then (i) is satisfied if and only if \( c_{ij}^{*} \geq c_{k,l}^{*} \), implying that \( T^* \) is a maximum spanning tree with \( c_{ij}^{*} \) as arc lengths.

(b) Let \( P_{k,l} \) be the unique path in the tree \( T^k \) from node \( k \) to node \( l \). For each nontree arc \((k, l)\) in \( T^k \), determine the interval \([\lambda_{k,l}^{*}, \lambda_{k,l}^{*}]\) for which the nontree arc \((k, l)\) continues to satisfy the path optimality conditions. It can be easily verified that \( \lambda_{k,l}^{*} = \min \{ (c_{ij}^{0} - c_{k,l}^{0})/(c_{ij}^{0} - c_{k,l}^{0}) : (i, j) \in P_{k,l} \} \) and \( \lambda_{k,l} = \min \{ c_{ij}^{*} : (i, j) \in P_{k,l} \} \). Then \( \lambda_{k,l} = \max \{ \lambda_{k,l}^{*} : (k, l) \notin T^k \} \) and \( \lambda = \min \{ \lambda_{k,l}^{*} : (k, l) \notin T^k \} \). Suppose that \( \lambda = \lambda_{pq}^{*} \) for some nontree arc \((p, q)\). At \( \lambda = \lambda_{pq}^{*} \), the cost of the arc \((p, q)\) exactly equals the cost of some tree
arc, say (u, v), and for \( \lambda \geq \lambda^* \), it is less. Replacing the tree arc (u, v) by the nontree arc (p, q) yields an alternate optimal spanning tree at \( \lambda = \lambda^* \), which is adjacent to \( T^{\lambda_*} \).

(c) We start with the maximum spanning tree with respect to \( c^*_ij \) (which is an optimal tree for \( \lambda = -\infty \)). We then use the method described in part (b) to obtain the largest value of \( \lambda = \lambda^* \) for which this spanning tree remains optimal.

Let \( \lambda^* = \lambda^* \), for some nontree arc (p, q), and arc (u, v) is the tree arc in the tree path from node p to node q whose cost equals that of the arc (p, q). Replacing the tree arc (u, v) by the nontree arc (p, q) gives us a new spanning tree. We repeat this process until \( \lambda^* = \infty \).

**EXERCISE 13.37.** This algorithm is exactly the same as the algorithm used for the minimum cost-to-time ratio cycle problem (described in Section 5.7), except that in this case we compute the minimum spanning trees after defining the length of arc (i, j) as \( l_{ij} = c_{ij} - \mu \tau_{ij} \). If the cost of the minimum spanning tree is negative, positive, or zero, then \( \mu^* < \mu \), \( \mu^* > \mu \) or \( \mu^* = \mu \), respectively, where \( \mu^* \) is the optimal ratio. The initial interval for search is \([-C, C]\) and we apply binary search until the width of the interval is less than \( 1/\tau_o^2 \), where \( \tau_o = \max\{\tau_{ij} : (i, j) \in A\} \).

Clearly, this approach applies a minimum spanning tree algorithm \( O(\log(\tau_o C)) \) and thus runs in polynomial time.

**EXERCISE 13.39.** (a) A l-forest is clearly an independence system. Suppose \( F \) and \( F^1 \) are two l-forests and \( |F^1| > |F| \geq 1 \). If \( F \) is a forest, then one can add any arc (i,j) \( \in F^1 - F \) to \( F \) and obtain a larger l-forest. So we assume that \( F \) is not a forest. Then for some arcs (u,v) \( \in F \) and (i,j) \( \in F^1\), \( F - \{(u,v)\} \) and \( F^1 - \{(i,j)\} \) are forests. (Note: If \( F^1 \) is a forest, we can choose (i,j) as any of its arcs.) Since forests are matroids (see Exercise 13.41) and \( F^1 - \{(i,j)\} \succ F - \{(u,v)\} \), we can add some arc (p,q) from \( F^1 - \{(i,j)\} \) to \( F - \{(u,v)\} \) so that \( F \cup \{(p,q)\} \) is a forest. We know that (p,q) \( \neq (u,v) \) since \( F \) is not a forest. Therefore, \( F \cup \{(p,q)\} \) is a l-forest, and thus l-forests satisfy the matroid growth property.

**EXERCISE 13.41.** Recall that a tree \( T \) with \( p \) nodes contains \( p-1 \) arcs, that is, exactly \( p \) nodes are adjacent to some arc of \( T \). The forest \( F_p \) decomposes into a collection of trees \( T_1, T_2, \ldots, T_k \) with \( n_1, n_2, \ldots, n_k \) nodes. At most \( (n_j - 1) \) arcs of \( F_{p+1} \) can have both their endpoints in the tree \( T_j \) (since \( F_{p+1} \) contains no cycles and \( T_j \) has \( n_j \) nodes). But since \( \sum_{1 \leq j \leq k} (n_j - 1) = p \), some arc (u,v) in \( F_{p+1} \) must connect two different trees \( T_i \) and \( T_j \). Adding arc (u,v) to \( F_p \) produces a forest with \( p+1 \) arcs.

**EXERCISE 13.43*.** Note: Constraint (13.3b) should read “for all \( S \subseteq E \).”

(a) Let \( y \) denote the 0-1 incidence vector of a basis \( B \). Then \( \sum_{e \in E} y_e = |B| = r(E) \) by definition of a basis, \( \sum_{e \in S} y_e = |S \cap B| \). Since \( S \cap B \) is an independent set within \( S \), \( |S \cap B| \leq r(S) \). Therefore, \( y \) is a feasible solution to the system (13.3a) - (13.3c).

(b) For the minimum spanning tree on a graph \( G=(N,E) \) with \( n \) nodes, \( r(E)=n-1 \). Let \( Q \) be any set of nodes in \( G \) and let \( S \) be the set of edges with both endpoints in \( Q \). \( r(S) \) is the largest sized forest in the graph \( (Q,S) \). Therefore, \( r(S) \leq |Q| - 1 \). So (13.3b) either contains every constraint in the formulation (13.2) or contains a constraint that is stronger (e.g., has a smaller right-hand side).
(c) Let $e_1, e_2, \ldots, e_p$, with $p = r(E)$, be the elements of $E$ chosen, in order, by the greedy algorithm and let $w_j$ denote the weight of $e_j$. Let $S_j$ be the span of the elements $e_1, e_2, \ldots, e_j$, that is, $S_j$ is the set of all elements of $E$ that are dependent upon the elements $e_1, e_2, \ldots, e_j$. Set

$$
\mu_{S_j} = w_{j+1} - w_j \geq 0 \quad \text{for } j = 1, 2, \ldots, j-1
$$

$$
\mu_N = -w_p
$$

and $\mu_S = 0$ otherwise.

The reduced cost $w^\mu_e$ of any element $e$ is given by $w^\mu_e = w_e + \sum_{S \supseteq e} \mu_S$. For each $j=1, 2, \ldots, k$, $w^\mu_e = w_j + (w_{j+1} - w_j) + (w_{j+2} - w_{j+1}) + \ldots + (w_p - w_{p-1}) - w_p = 0$.

If $e \neq e_j$ for all $j=1, 2, \ldots, p$, let $k$ be the minimum index satisfying the property that $e$ belongs to $S_k$. Then,

$$
w^\mu_e = w^\mu_{e_k} + (w_e - w_k) \geq 0.
$$

Consequently, if we set $x_{e_j} = 1$ for $j=1, 2, \ldots, p$ and $x_e = 0$ otherwise, the integer vector $x$ satisfies the linear programming complementary slackness conditions with respect to the dual variables $\mu_S$. Therefore, $x$ solves the linear programming (13.3) and the associated integer program formed by restricting each $x_e$ to be 0 or 1. (Note: The procedure specified in this solution is slightly different than the proof we have given for Theorem 13.9 since it sets non-zero multipliers on the forest determined by the arcs $e_1, e_2, \ldots, e_j$ at each step and not on the component determined by the last element $e_j$).

**EXERCISE 13.45.** Let $c_e = -1$ if $e \in \mathcal{I}$

$$
c_e = -1 + \frac{1}{|\mathcal{I}|+1} \quad \text{if } e \in \mathcal{I} - \mathcal{I}
$$

$$
c_e = -1 \quad \text{if } e \notin \mathcal{I} \cup \mathcal{I}.
$$

Then the greedy algorithm selects the set $\mathcal{I}$. But,

$$
c(\mathcal{I}) = |\mathcal{I}| + \frac{1}{|\mathcal{I}|+1} \quad \text{if } \mathcal{I} \leq \mathcal{I} - |\mathcal{I}| + \frac{1}{|\mathcal{I}|+1} \quad \text{if } |\mathcal{I}| < |\mathcal{I}| + 1 \leq |\mathcal{I}| = c(\mathcal{I}).
$$

$\mathcal{I}$ is optimal, yet the greedy algorithm terminates with $\mathcal{I}$. 