EXERCISE 8.1. (a) The flow decomposition algorithm presented in Section 3.5 will always decompose a circulation in a unit capacity network into unit flows along arc-disjoint directed cycles.

(b) The flow decomposition algorithm presented in Section 3.5 will always decompose a circulation in a simple network into unit flows along node-disjoint directed cycles.

EXERCISE 8.3. (a) Consider the graph $G'$ obtained as follows. Add node $s$ and arcs $(s,k)$ and $(s,l)$, add node $t$ plus arcs $(p,t)$ and $(q,t)$ and delete arc $(k,l)$. Now there are two cases. Case 1. There exist two node-disjoint paths from node $s$ to node $t$. then we have the situation shown in Figure S8.3. Clearly, $P_1 \cup P_2 \cup (k,l)$ is a path from node $p$ to node $q$ passing through the arc $(k,l)$. Case 2. There do not exist two node-disjoint paths from node $s$ to node $t$, but there is one node-disjoint path from node $s$ to node $t$. Notice that in this case there exists a node whose deletion disconnects the original network $G$ into two parts contradicting that the network is biconnected.

(b) If $q$ is connected to both $p$ and $r$, then $p,q,r$ is the path. If $q$ is connected to some other node $w$, then by part (a) there is a path from $p$ to $r$ passing through the arc $(q,w)$.

(c) There are at least two node-disjoint paths from $p$ to $r$. At least one of these does not contain $q$.

EXERCISE 8.5. Let us assume that the arc adjacency list of each node is arranged in the increasing order of the head nodes. The algorithm will terminate within the first phase itself after augmenting flows along the paths 1-5-6-9 (of length 3 units) and 1-4-5-8-9 (of length 4 units).

EXERCISE 8.7. The unit capacity maximum flow algorithm will still solve the problem in $O(\min\{n^{2/3}m, m^{3/2}\})$ time in the case when only the source and sink nodes have arcs with arbitrary capacities incident on them. (The complexity proof for the unit capacity maximum flow problem holds in this case as well.) In the case when one other node (say $i$) has arcs with arbitrary capacities incident on it, the complexity of the first phase of the algorithm is $O(\min\{n^{2/3}m, m^{3/2}\})$ and the complexity of the second phase of the algorithm is $O(m[u+\min\{n^{2/3}, m^{1/2}\}])$, where $u = \min\{\sum_{j:(i,j) \in A} u_{ij}, \sum_{j:(j,i) \in A} u_{ji}\}$. The complexity of the second phase follows from the fact that a maximum of $u + \min\{n^{2/3}, m^{1/2}\}$ flow may be pushed through the node layer containing node $i$, if it is one of the two layers immediately separated by the minimum cut.

EXERCISE 8.9. When the preflow-push algorithm is applied to a unit capacity network, then each push is a saturating push. Therefore, there will not be any nonsaturating pushes. If we examine the worst-case analysis of the preflow-push algorithm, we find that the running time is $O(nm + n^2m)$ where $O(n^2m)$ term is contributed by the
nonsaturating pushes and the remaining operation takes $O(nm)$ time. In the absence of the nonsaturating pushes, the preflow-push algorithm will run in $O(nm)$ time.

**Exercise 8.11** This exercise is incorrectly stated. The correct statement is: "Let $x$ be a flow in a directed network $G$; assume that $x$ is not a maximum $s$-$t$ flow. Let $P$ and $P'$ denote two successive shortest paths (i.e., $P'$ is the shortest path after augmentation on path $P$) and suppose that $P'$ contains at least one arc whose reversal lies on $P$. Show that $|P'| \geq |P| + 2$.

Suppose that arc $(p, q)$ is a forward arc in the path $P$ and $(q, p)$ is a forward arc in $P'$. Suppose that in $P$ the length of the subpath from node $s$ to node $p$ is $\alpha$ (to node $q$ is $\alpha+1$), and the length of the subpath from node $q$ to node $t$ is $\beta$. As augmentations do not decrease the lengths of the shortest paths to any node, in $P'$, the length of the subpath from node $s$ to node $q$ is at least $\alpha+1$, and the length of the subpath from node $p$ to node $t$ is at least $\beta+1$.

**Exercise 8.13.** Construct a complete graph $G^O = (N_1 \cup N_2, A^O)$ where $N_1 = \{1, 2, \ldots, n\}$, $N_2 = \{1', 2', \ldots, n'\}$, and $A^O = \{(i, j) : i \in N_1, j \in N_2\}$. Set the supply of node $i$ in $N_1$ to $\alpha(i)$ and the demand of each node in $N_2$ to $\beta(j)$. A graph in which every node $i$ has outdegree equal to $\alpha(i)$ and indegree equal to $\beta(i)$ will exist if and only if a feasible flow exists in $G^O$. We can determine the presence or absence of such a feasible flow by solving a maximum flow problem.

**Exercise 8.15.** Assume that the source nodes lies in $N_2$ and the sink node lies in $N_1$. When the bipartite preflow-push algorithm is applied to the problem in Figure 8.21, the following sequence of operations is performed. The algorithm performs the following operations: (1) In the preprocessing step, saturate the arcs $(1, 2)$, $(1, 3)$, $(1, 4)$ and $(1, 5)$; (2) push 4 units of flow along 2-6-9; (3) push 1 unit of flow along 2-7-9; (3) push 6 units of flow along 3-7-9; (4) push 8 units of flow along 4-7-9; (5) push 1 unit of flow along 5-7-9; (6) push 8 units of flow along 5-8-9; (9) relabel node 5 as $d(5) = d(1) + 1$; (10) push 1 unit of flow along 5-1. Figure S8.15 shows the optimal flow.

**Exercise 8.17.** (a) Clearly, the number of nonsaturating pushes between two consecutive relabel operations in a bipartite network is at most $n_1$. Since, the total number of relabel operations is $O(n_1^2)$, the total number of nonsaturating pushes is $O(n_1^3)$. As each node in $N_1$ can be relabeled at most $O(n_1)$ times, the total number of saturating pushes is $O(n_1m)$. Hence the algorithm runs in $O(n_1^3 + n_1m)$ time.

(b) We use the potential function $\Phi = \max\{d(i) : i \text{ is active and } i \in N_1\}$. We note that in each phase of the bipartite FIFO preflow-push algorithm, no more than $n_1$ node examinations can take place, because every node which is
examined lies in $N_1$. Further, the number of relabel operations is bounded by $O(n_1^2)$. Now, use similar arguments as in Section 7.7 to show that the number of nonsaturating pushes is $O(n_1^3)$. Consequently, the running time of the FIFO implementation is also $O(n_1m + n_1^3)$.

(c) Consider the potential function $\Phi = \sum_{i \in N_1} e(i)d(i)/\Delta$. The increase in $\Phi$ due to node relabelings in each scaling phase (in fact, over all the scaling phases) is at most $O(n_1^2)$. Further, each nonsaturating push will decrease $\Phi$ by at least one unit. Hence the number of nonsaturating pushes in each scaling phase is $O(n_1^2)$. It follows that the total time taken by the algorithm (including saturating pushes and arc scannings) is $O(n_1m + n_1^2\log U)$.

**Exercise 8.19.** Let the faces of the graph be labeled $f_1, f_2, \ldots, f_k$. Let $a_i$ denote the number of arcs in the $i$th face. Then $2m \geq \sum_i a_i$ since each arc appears in at most two faces.

(a) Consider the graph in Figure 8.6(a). Each face has at least 4 arcs since each face is a cycle with more than two arcs, and all cycles are have an even number of arcs. By Euler’s formula the number of faces is $f = m-n+2 = 9 - 6 + 2 = 5$. We conclude that $2m = 18 \geq \sum_i a_i \geq 5 \times 4 = 20$, which is a contradiction.

(b) Now consider the graph in Figure 8.6(b). Each face has at least 3 arcs, and by Euler’s formula the number of faces is $f = m-n+2 = 10 - 5 + 2 = 7$. We conclude that $2m = 20 \geq \sum_i a_i \geq 21$, which is a contradiction.

**Exercise 8.21.** The distance labels in the dual network are illustrated in Figure S8.21(a) while the maximum flow is illustrated in Figure S8.21(b). The path $1^*-3^*-4^*-8^*-7^*-10^*$ is a shortest path in the dual network and it corresponds to the following minimum cut in the dual network: $\{(4, 7), (3, 7), (3, 6), (2, 6), (2, 5)\}$. 
**Exercise 8.23.** Let $G^*$ be the dual of the network $G$. Now, every directed $s$-$t$ path in $G$ corresponds to some $s^*$-$t^*$ cut in $G^*$, and hence the minimum number of arcs in a directed path from $s$ to $t$ is equal to the minimum number of arcs in any $s^*$-$t^*$ cut in $G^*$. Now every arc in the path from $s^*$ to $t^*$ in $G^*$ has a (uniquely) corresponding arc in $G$. Hence, the maximum number of arc-disjoint $s$-$t$ cuts in $G$ is equal to the maximum number of arc-disjoint $s^*$-$t^*$ paths in $G^*$, which by Theorem 6.7 is again equal to the minimum number of arcs in any $s^*$-$t^*$ cut in $G^*$.

**Exercise 8.25.** Note that this exercise contains a typographic error and the maximum flow in the network has value 3, not 4 as stated. Figure S8.25 shows the initial arc flows and initial distance labels, and Figure 8.11(c) in the book shows the initial dynamic tree data structure. The algorithm performs the following sequence of dynamic tree operations before discovering an augmenting path.

(i) cut (8), cut(6), cut(5), cut(2) and cut(1).
(ii) link(1, 3, 2), link(5, 7, 2), link(9, 11, 2), and link(11, 13, 2) by the procedure tree-advance. Figure S8.25(b) shows the dynamic tree data structure at this stage.
(iii) augment 1 unit of flow along the path 1-3-5-7-9-11-13-15-16. This augmentation saturates the arc (13, 15).
(iv) perform cut(13) which is followed by tree-retreat(13). This operation increases the distance label of node 13 to 4. If we simultaneously maintain the numb array (as described in Section 7.4), then we find that numb(2) = 0 and terminate the algorithm. Figure 8.25(c) gives the maximum flow solution.
EXERCISE 8.27. If we consider a flow decomposition of the maximum pseudoflow, then the flow can be decomposed into (i) path flows from node \( s \) to node \( t \); (ii) path flows from source to excess nodes; and (iii) flows around cycles. If we eliminate all but the flows from node \( s \) to \( t \), then we get a maximum flow. So it remains to show how one can determine a flow decomposition in \( O(m \log n) \) time using dynamic trees.

We use the following algorithm to convert the maximum preflow into a maximum flow. Select an excess node \( j \), identify an incoming arc \((i, j)\) into node \( j \) with positive flow. Next select node \( i \) and identify an incoming arc \((k, i)\) into node \( i \) with positive flow. Repeat this iteration until either we discover a directed cycle or reach the source node. We reduce the flow on the cycle or the path so that some arc flow becomes zero. We repeat this process until there is no node with positive excess. It is easy to see that this algorithm can be implemented in \( O(m^2) \) time. Using the dynamic tree data structure, this algorithm can be implemented in \( O(m \log n) \) time.

We first construct the network \( G' = (N, A') \) in the following manner: for each arc \((i, j) \in A\) if \( x_{ij} > 0 \) then add an arc \((j, i)\) to \( A' \) with the residual capacity \( r_{ji} = x_{ij} \). The algorithm is a simplified version of the shortest augmenting path algorithm described in Section 8.5. The algorithm has the following features: (i) each arc with a positive residual capacity is admissible; (ii) the dynamic tree is a collection of rooted trees; (iii) the algorithm performs tree-advances to identify a path from an excess node to the source node or discovers a cycle; (iv) it performs a tree-augment to send the maximum possible flow; (v) it cuts the saturated arc and repeats the process; (vi) the algorithm does not perform the tree-retreat operation because each excess node has an outgoing arc, and whenever a node (other than the source node) has an incoming arc it must have an outgoing arc too. Each iteration of the algorithm takes \( O(\log n) \) amortized time per iteration and each iteration saturates at least one arc. Hence there will be at most \( m \) iterations and the algorithm would run in \( O(m \log n) \) time.

EXERCISE 8.29#. In part (b) of the exercise, the factor of \( 2 \) from \( \lceil 2m/n \rceil \) has been inadvertently missed.
(a) It is easy to see that $\alpha[i, j] = \alpha[j, i]$ because any path in an undirected network from node $i$ to node $j$ is also a path from node $j$ to node $i$, and vice-versa.

(b) If $i$ be the node having the minimum degree, then removal of all the arcs emanating from node $i$ disconnects it from the network. Since the sum of the degrees of the nodes in the network is equal to $2m$, the minimum degree of any node is not larger than $\lceil 2m/n \rceil$, i.e., $\alpha(G) \leq \lceil 2m/n \rceil$.

(c) Suppose that the minimum disconnecting set containing the subset $A^0$ of arcs partitions the network into the components $N_1, N_2, \ldots, N_k$, $k \geq 3$. The disconnecting set must contain some arc $(i, j)$ in $[N_1, \bar{N}_1]$ where $\bar{N}_1 = N_2 \cup N_3 \cup \ldots \cup N_k$. Without any loss of generality, we may assume that $j \in N_2$. Then, if we remove arc $(i, j)$ from the minimum disconnecting set $A^0$, then the remaining set of arcs still disconnects the network into the components containing the node sets $N_1 \cup N_2$, $N_3$, ..., $N_k$, and its cardinality is one less, which contradicts that $A^0$ is a minimum disconnecting set. This establishes that any minimum disconnecting set cannot partition the network into more than two components.

(d) It is obvious that removal of any arc from the spanning tree partitions the network into two components. Hence, the arc connectivity of a spanning tree equals 1.

(e) It is obvious that removal of any two arcs from a cycle partitions it into two components, and removal of any one arc results in a path containing all the nodes of the cycle. Hence, arc connectivity of a cycle equals 2.

**EXERCISE 8.31.** The network illustrated in Figure S8.31 has only two arc-disjoint paths from node 1 to node 2, but each node has a degree of three.

![Figure S8.31](image)

**EXERCISE 8.33.** Let $p$ be a node with the minimum outdegree. Clearly, $\delta \leq \lceil m/n \rceil$. Further, let node $[S, S^-]$ be a minimum disconnecting set. Then node $p \in S$ or $p \in S^-$. Consequently, we solve a maximum flow problem from node $p$ to every other node (with each arc capacity equal to 2), and a maximum flow problem from every other node to $p$. The cut of minimum value among these $2n$ minimum cuts is a minimum disconnecting set. To solve a maximum flow problem takes $O(\min(n^{2/3}m, m^{3/2}))$ time, giving a time bound of $O(\min(n^{5/3}m, nm^{3/2}))$ for determining a minimum disconnecting set. Alternatively, we can use the labeling algorithm to solve the maximum flow problem, which takes a total of $O(m^2)$ time (the arguments for the undirected case apply for the directed case too). Hence, we can determine the arc connectivity of a directed network in $O(\min(n^{5/3}m, nm^{3/2}, m^2))$ time.

**EXERCISE 8.35**. In part (d) of the exercise, we fix any arbitrary node, not necessarily a node with the minimum degree.

(a) If $(i, j) \notin A$, then even if we delete all the nodes in $N_{-i, j}$, the nodes $i$ and $j$ will remain connected by the arc $(i, j)$. Hence $\beta[i, j]$ is not defined.

(b) If we delete all the neighbors of a given node $i$, then node $i$ is disconnected from the rest of the network. Hence, the node connectivity is no larger than the minimum degree of any node in the graph. But the minimum degree of
any node in the network \( \leq \frac{\text{sum of degrees}}{\text{number of nodes}} \), i.e., node-connectivity \( \leq \) minimum degree of any node \( \leq 2\lfloor m/n \rfloor \).

(c) Let \( [S^*, S^-] \) be a minimum disconnecting arc set in the network. For each arc \((i, j)\) in \([S^*, S^-]\) if we delete node \(i\), all the arcs in \([S^*, S^-]\) will also get deleted. Observe that some of the end points of the arcs in \([S^*, S^-]\) may be common, and we may not have deleted as many nodes as the number of arcs in \([S^*, S^-]\). Hence a node disconnecting set exists, whose cardinality is no larger than that of the minimum arc disconnecting set. Consequently, the node connectivity of a network is no more than its arc connectivity.

(d) We can determine the node connectivity of an algorithm using the following method. Fix any node \(p \in N\) (not necessarily a node with minimum degree), and determine \(\beta[p, j]\) for each \(j\) for which \((p, j) \in A\). The minimum of these numbers is the node connectivity of the graph. If we apply this method to the graph shown in Figure 8.25 with \(p = 2\), then we will find that \(\beta[4, j] = 2\) for each \(j\). Hence, this method will declare the node connectivity of the network equal to 2. But observe that when we delete node 4 from the network, it becomes disconnected implying that the node connectivity of the network is indeed one.