Property 7.2. *If $d(s) \geq n$, the residual network contains no directed path from the source node to the sink node.*

The correctness of this observation follows from the facts that $d(s)$ is a lower bound on the length of the shortest path from s to t in the residual network, and therefore no directed path can contain more than $(n - 1)$ arcs. Therefore, if $d(s) \geq n$, the residual network contains no directed path from node s to node t.

We now introduce some additional notation. We say that the distance labels are *exact* if for each node i, $d(i)$ equals the length of the shortest path from node i to node t in the residual network. For example, in Figure 7.1, if node 1 is the source node and node 4 is the sink node, then $d = (0, 0, 0, 0)$ is a valid vector of distance label, and $d = (3, 1, 2, 0)$ is a vector of exact distance labels. We can determine exact distance labels for all nodes in $O(m)$ time by performing a backward breadth-first search of the network starting at the sink node (see Section 3.4).

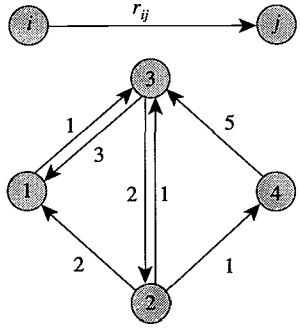

Figure 7.1 Residual network.

Admissible Arcs and Admissible Paths

We say that an arc (i, j) in the residual network is *admissible* if it satisfies the condition that $d(i) = d(j) + 1$; we refer to all other arcs as *inadmissible*. We also refer to a path from node s to node t consisting entirely of admissible arcs as an *admissible path*. Later, we use the following property of admissible paths.

Property 7.3. *An admissible path is a shortest augmenting path from the source to the sink.*

Since every arc (i, j) in an admissible path P is admissible, the residual capacity of this arc and the distance labels of its end nodes satisfy the conditions (1) $r_{ij} > 0$, and (2) $d(i) = d(j) + 1$. Condition (1) implies that P is an augmenting path and condition (2) implies that if P contains k arcs, then $d(s) = k$. Since $d(s)$ is a lower bound on the length of any path from the source to the sink in the residual network (from Property 7.1), the path P must be a shortest augmenting path.

7.3 CAPACITY SCALING ALGORITHM

We begin by describing the maximum capacity augmenting path algorithm and noting its computational complexity. This algorithm always augments flow along a path with the maximum residual capacity. Let x be any flow and let v be its flow value.

As before, let v^* be the maximum flow value. The flow decomposition property (i.e., Theorem 3.5), as applied to the residual network $G(x)$, implies that we can find m or fewer directed paths from the source to the sink whose residual capacities sum to $(v^* - v)$. Thus the maximum capacity augmenting path has residual capacity at least $(v^* - v)/m$. Now consider a sequence of $2m$ consecutive maximum capacity augmentations starting with the flow x. If each of these augmentations augments at least $(v^* - v)/2m$ units of flow, then within $2m$ or fewer iterations we will establish a maximum flow. Note, however, that if one of these $2m$ consecutive augmentations carries less than $(v^* - v)/2m$ units of flow, then from the initial flow vector x, we have reduced the residual capacity of the maximum capacity augmenting path by a factor of at least 2. This argument shows that within $2m$ consecutive iterations, the algorithm either establishes a maximum flow or reduces the residual capacity of the maximum capacity augmenting path by a factor of at least 2. Since the residual capacity of any augmenting path is at most $2U$ and is at least 1, after $O(m \log U)$ iterations, the flow must be maximum. (Note that we are essentially repeating the argument used to establish the geometric improvement approach discussed in Section 3.3.)

As we have seen, the maximum capacity augmentation algorithm reduces the number of augmentations in the generic labeling algorithm from $O(nU)$ to $O(m \log U)$. However, the algorithm performs more computations per iteration, since it needs to identify an augmenting path with the maximum residual capacity, not just any augmenting path. We now suggest a variation of the maximum capacity augmentation algorithm that does not perform more computations per iteration and yet establishes a maximum flow within $O(m \log U)$. Since this algorithm scales the arc capacities implicitly, we refer to it as the *capacity scaling algorithm.*

The essential idea underlying the capacity scaling algorithm is conceptually quite simple: We augment flow along a path with a *sufficiently large* residual capacity, instead of a path with the maximum augmenting capacity because we can obtain a path with a sufficiently large residual capacity fairly easily—in $O(m)$ time. To define the capacity scaling algorithm, let us introduce a parameter Δ and, with respect to a given flow x, define the *Δ-residual network* as a network containing arcs whose residual capacity is at least Δ. Let $G(x, \Delta)$ denote the Δ-residual network. Note that $G(x, 1) = G(x)$ and $G(x, \Delta)$ is a subgraph of $G(x)$. Figure 7.2 illustrates this definition. Figure 7.2(a) gives the residual network $G(x)$ and Figure 7.2(b) gives the Δ-residual network $G(x, \Delta)$ for $\Delta = 8$. Figure 7.3 specifies the capacity scaling algorithm.

Let us refer to a phase of the algorithm during which Δ remains constant as a *scaling phase* and a scaling phase with a specific value of Δ as a *Δ-scaling phase.* Observe that in a Δ-scaling phase, each augmentation carries at least Δ units of flow. The algorithm starts with $\Delta = 2^{\lfloor \log U \rfloor}$ and halves its value in every scaling phase until $\Delta = 1$. Consequently, the algorithm performs $1 + \lfloor \log U \rfloor = O(\log U)$ scaling phases. In the last scaling phase, $\Delta = 1$, so $G(x, \Delta) = G(x)$. This result shows that the algorithm terminates with a maximum flow.

The efficiency of the algorithm depends on the fact that it performs at most $2m$ augmentations per scaling phase. To establish this result, consider the flow at the end of the Δ-scaling phase. Let x' be this flow and let v' denote its flow value. Furthermore, let S be the set of nodes reachable from node s in $G(x', \Delta)$. Since