size of the map N. Furthermore, a naive implementation of data association may result in evaluating the measurement likelihood for each of the N features in the map, resulting again in linear complexity in N. We note that a poor implementation of the sampling process might easily add another $\log N$ to the update complexity.

Efficient implementations of FastSLAM require only $O(M \log N)$ update time. This is logarithmic in the size of the map N. First, consider the situation with known data association. Linear copying costs can be avoided by introducing a data structure for representing particles that allow for more selective updates. The basic idea is to organize the map as a *balanced binary tree*. Figure 13.8a shows such a tree for a single particle, in the case of $N = 8$ features. Notice that all Gaussian parameters $\mu_j^{[k]}$ and $\Sigma_j^{[k]}$ for all j are located at the leaves of the tree. Assuming that the tree is approximately balanced, accessing a leaf requires time logarithmic in N.

Suppose FastSLAM incorporates a new control u_t and a new measurement z_t. Each new particle in Y_t will differ from the corresponding one in Y_{t-1} in two ways: First, it will possess a different pose estimate obtained via (13.26), and second, the observed feature's Gaussian will have been updated, as specified in Equations (13.47) and (13.48). All other Gaussian feature estimates, however, will be equivalent to the generating particle. When copying the particle, thus, only a single path has to be modified in the tree representing all Gaussians, leading to the logarithmic update time.

An illustration of this "trick" is shown in Figure 13.8b: Here we assume $c_t^i = 3$, hence only the Gaussian parameters $\mu_3^{[k]}$ and $\Sigma_3^{[k]}$ are updated. Instead of generating an entire new tree, only a single path is created, leading to the Gaussian $c_t^i = 3$. This path is an incomplete tree. The tree is completed by copying the missing pointers from the tree of the generating particle. Thus, branches that leave the path will point to the same (unmodified) subtree as that of the generating tree. Clearly, generating this tree takes only time logarithmic in N. Moreover, accessing a Gaussian also takes time logarithmic in N, since the number of steps required to navigate to a leaf of the tree is equivalent to the length of the path (which is by definition logarithmic). Thus, both generating and accessing a partial tree can be done in time $O(\log N)$. Since in each updating step M new particles are created, an entire update requires time in $O(M \log N)$.

Organizing particles in trees raises the question as to when to deallocate memory. Memory deallocation can equally be implemented in amortized logarithmic time. The idea is to assign a variable to each node—internal