8.7 An Application

An example of the use of the union/find data structure is the generation of mazes, such as the one shown in Figure 8.19. In Figure 8.19, the starting point is the top-left corner, and the ending point is the bottom-right corner. We can view the maze as a 50-by-88 rectangle of cells in which the top-left cell is connected to the bottom-right cell, and cells are separated from their neighboring cells via walls.

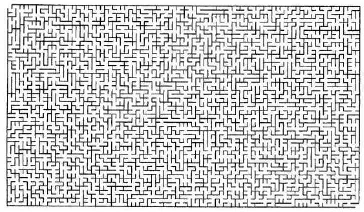
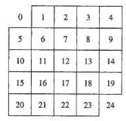


Figure 8.19 A 50-by-88 maze

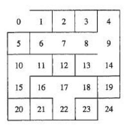


{0} {1} {2} {3} {4} {5} {6} {7} {8} {9} {10} {11} {12} {13} {14} {15} {16} {17} {18} {19} {20} {21} {22} {23} {24}

Figure 8.20 Initial state: all walls up, all cells in their own set

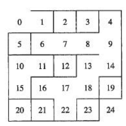
A simple algorithm to generate the maze is to start with walls everywhere (except for the entrance and exit). We then continually choose a wall randomly, and knock it down if the cells that the wall separates are not already connected to each other. If we repeat this process until the starting and ending cells are connected, then we have a maze. It is actually better to continue knocking down walls until every cell is reachable from every other cell (this generates more false leads in the maze).

We illustrate the algorithm with a 5-by-5 maze. Figure 8.20 shows the initial configuration. We use the union/find data structure to represent sets of cells that are connected to each other. Initially, walls are everywhere, and each cell is in its own equivalence class.



{0, 1} {2} {3} {4, 6, 7, 8, 9, 13, 14} {5} {10, 11, 15} {12} {16, 17, 18, 22} {19} {20} {21} {23} {24}

Figure 8.21 At some point in the algorithm: several walls down, sets have merged; if at this point the wall between 8 and 13 is randomly selected, this wall is not knocked down, because 8 and 13 are already connected

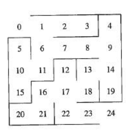


{0, 1} {2} {3} {4, 6, 7, 8, 9, 13, 14, 16, 17, 18, 22} {5} {10, 11, 15} {12} {19} {20} {21} {23} {24}

Figure 8.22 Wall between squares 18 and 13 is randomly selected in Figure 8.21; this wall is knocked down, because 18 and 13 are not already connected; their sets are merged

Figure 8.21 shows a later stage of the algorithm, after a few walls have been knocked down. Suppose, at this stage, the wall that connects cells 8 and 13 is randomly targeted. Because 8 and 13 are already connected (they are in the same set), we would not remove the wall, as it would simply trivialize the maze. Suppose that cells 18 and 13 are randomly targeted next. By performing two `find` operations, we see that these are in different sets; thus 18 and 13 are not already connected. Therefore, we knock down the wall that separates them, as shown in Figure 8.22. Notice that as a result of this operation, the sets containing 18 and 13 are combined via a `union` operation. This is because everything that was connected to 18 is now connected to everything that was connected to 13. At the end of the algorithm, depicted in Figure 8.23, everything is connected, and we are done.

The running time of the algorithm is dominated by the union/find costs. The size of the union/find universe is equal to the number of cells. The number of `find` operations is



{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24}

Figure 8.23 Eventually, 24 walls are knocked down; all elements are in the same set

proportional to the number of cells, since the number of removed walls is one less than the number of cells, while with care, we see that there are only about twice the number of walls as cells in the first place. Thus, if N is the number of cells, since there are two finds per randomly targeted wall, this gives an estimate of between (roughly) $2N$ and $4N$ `find` operations throughout the algorithm. Therefore, the algorithm's running time can be taken as $O(N \log^* N)$, and this algorithm quickly generates a maze.