Back–tracking and Branch–and–Bound

Backtracking

- Principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates
- If a partially constructed solution can be developed further without violating the problem's constraints, then the first remaining legitimate option for the next component is taken

Backtracking

- If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered.
- In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

n–Queens Problem

- Place n queens on an n × n chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagonal
- For n = 1, the problem has a trivial solution, and it is easy to see that there is no solution for n = 2 and n = 3
- Let us consider the four–queens problem and solve it by the backtracking technique

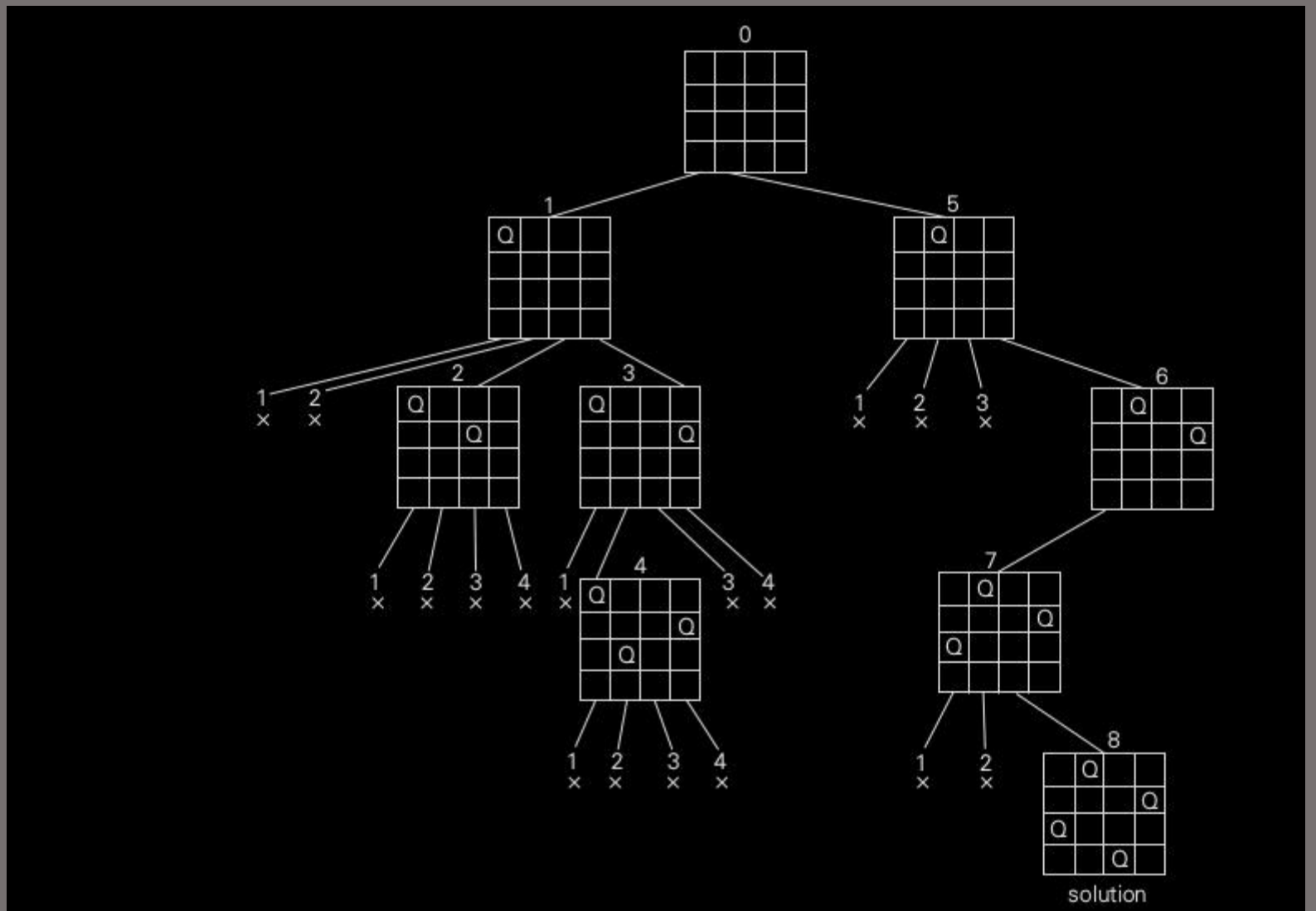

FIGURE 12.2 State-space tree of solving the four-queens problem by backtracking. × denotes an unsuccessful attempt to place a queen in the indicated column. The numbers above the nodes indicate the order in which the nodes are generated.

n–Queens Problem Recursive

```
n–queens(board, n, curr_row)
   if(curr_row == n) print_board and return
   for i to n
         if board[curr_row][i] is not attacked
                  board[curr_row][i] = 1
                  n–queens(board, n, curr_row+1)
                  // backtrack by removing queen placed in column i
                  // and try next position
                  board[curr_row][i] = 0
```

n–Queens Problem Iterative

```
n–queens(board, n, curr_row,  pos)
   if(curr_row==n) print board and return
   if sizeof(pos)<curr_row then i = 1 and
                        i = top(pos) +1, board[curr_row][top(pos)] = 0 otherwise
   for i to n
         if board[curr_row][i] is not attacked
                  push i into pos
                  board[curr_row][i] = 1
                  n–queens(board, n, curr_row+1,pos), return
   n–queens(board, n, curr_row–1,pos)
```