Recursion

- **Example 1:** Fibonacci Numbers
  1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

```
public static long fib(int n)
{
        if (n <= 1)
                return n;
        else
                return fib(n-1) + fib(n-2);
}
```

- **Example 2:** Towers of Hanoi

Recursion

- **Example 1:** Fibonacci Numbers
  1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

```
public static long fib(int n)
{
        if (n <= 1)
                return n;
        else
                return fib(n-1) + fib(n-2);
}
```

- **Example 2:** Towers of Hanoi

Figure 2.11

Recursive calls that ***rabbit(7)*** generates

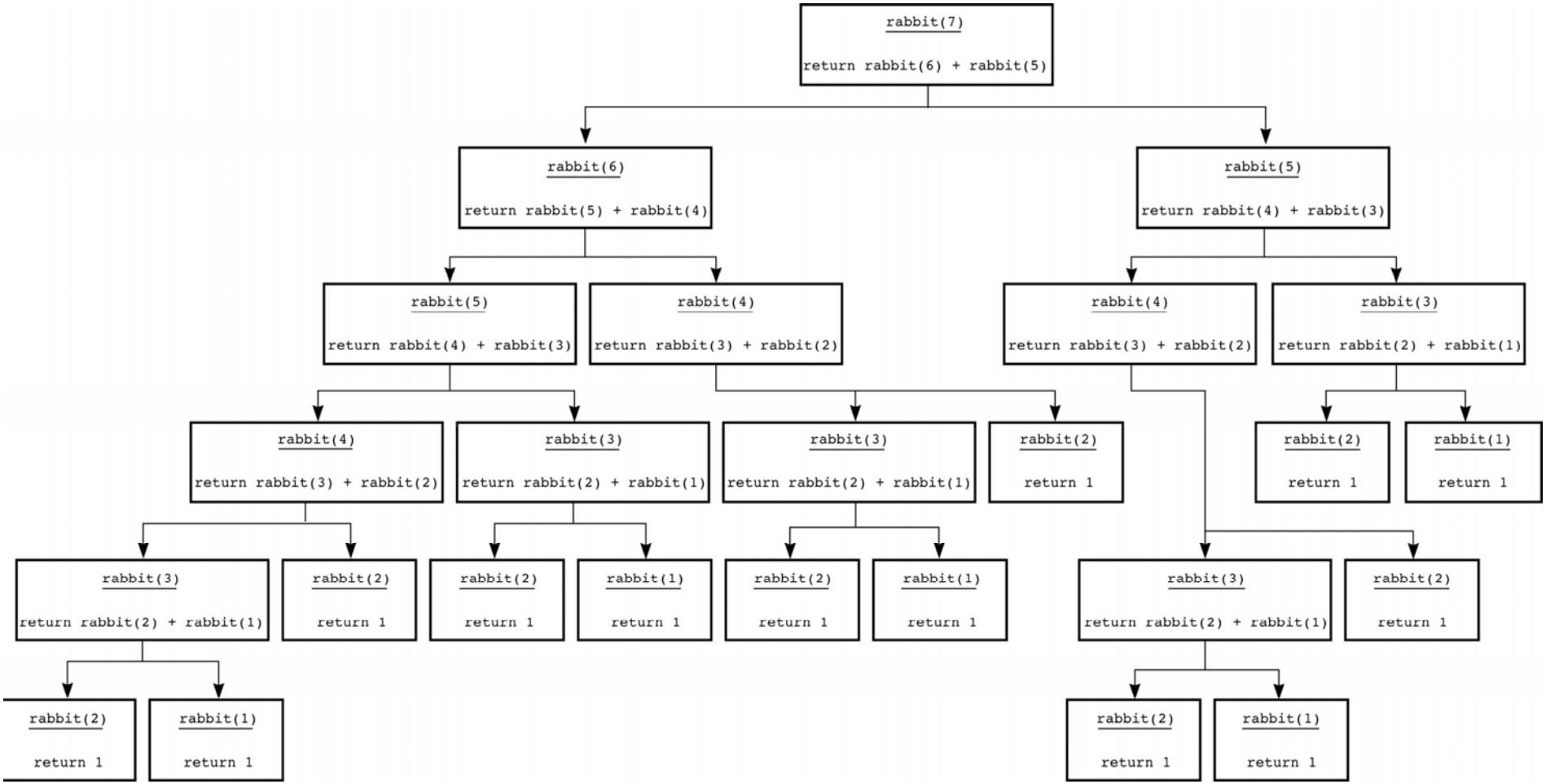


Figure 2.19a and b

a) The initial state; b) move *n* - 1 disks from *A* to *C*

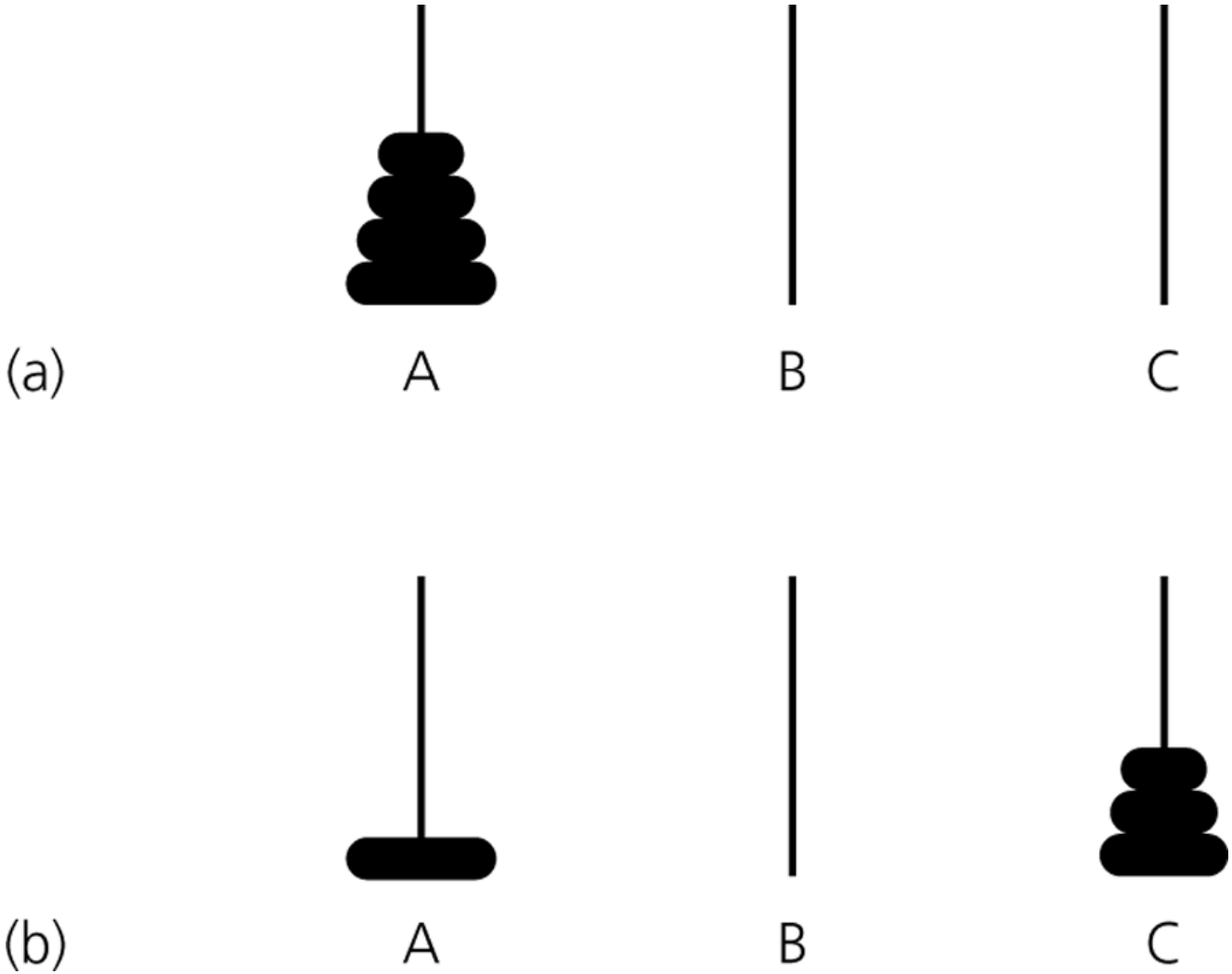
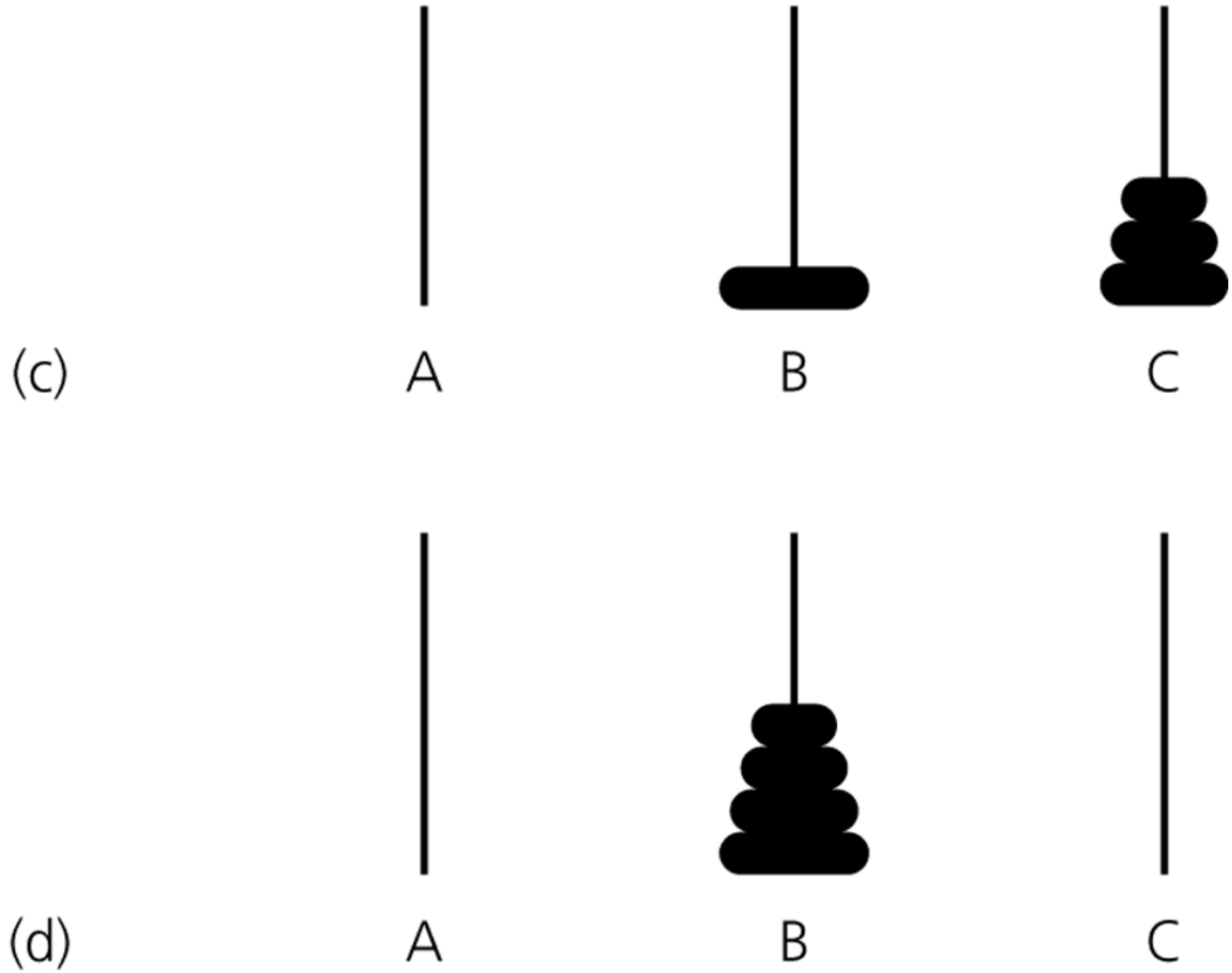


Figure 2.19c and d

c) move one disk from *A* to *B*; d) move *n* - 1 disks from *C* to *B*



Sample output

```
Move top disk from pole A to pole B
Move top disk from pole A to pole C
Move top disk from pole B to pole C
Move top disk from pole A to pole B
Move top disk from pole C to pole A
Move top disk from pole C to pole B
Move top disk from pole A to pole B
```

SolveTowers Solution

```
public static void solveTowers(int count, char source,
                               char destination, char spare)
{
  if (count == 1) {
    System.out.println("Move top disk from pole " + source +
                       " to pole " + destination);
  }
  else {
    solveTowers(count-1, source, spare, destination); // X
    solveTowers(1, source, destination, spare);       // Y
    solveTowers(count-1, spare, destination, source); // Z
  }  // end if
}  // end solveTowers
```

Figure 2.20

The order of recursive calls that results from ***solveTowers(3, A, B, C)***

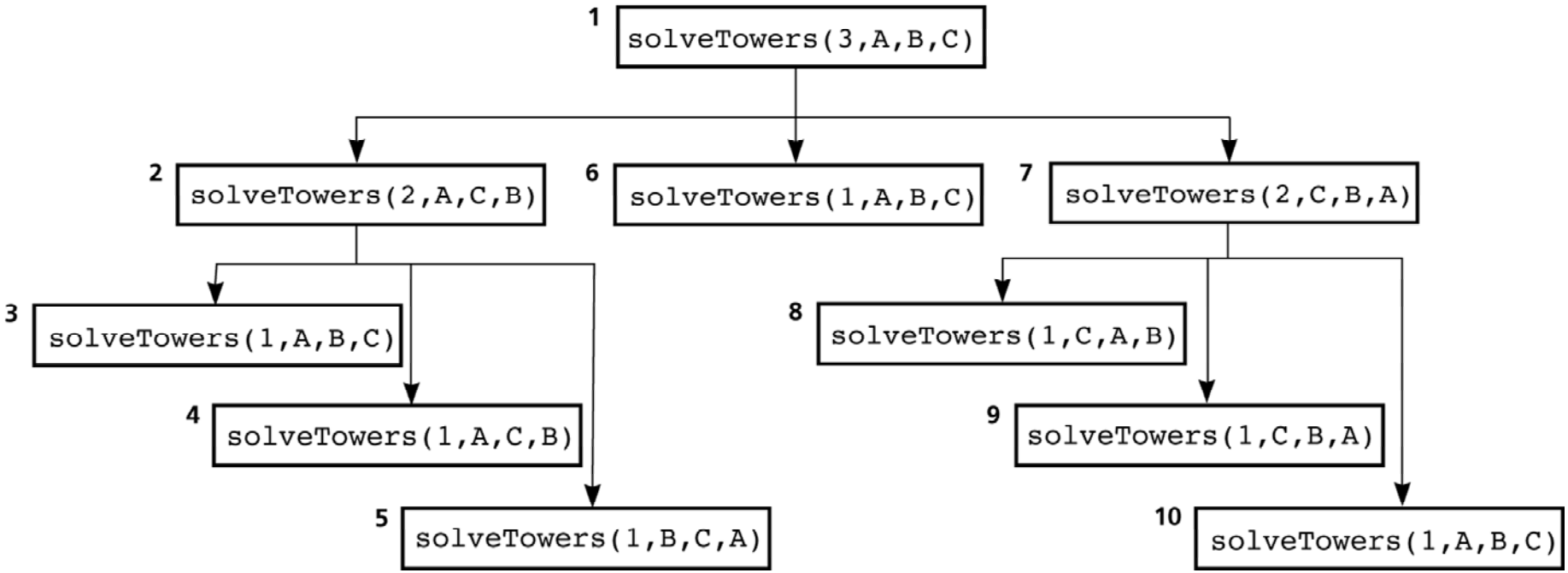
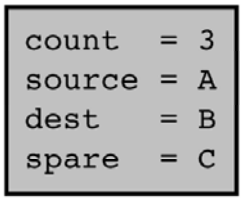


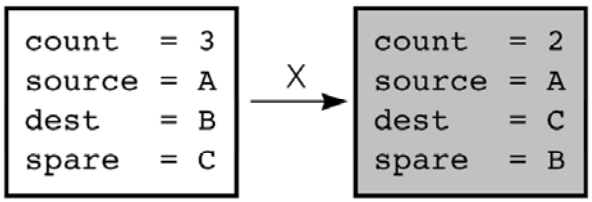
Figure 2.21a

Box trace of ***solveTowers(3, 'A', 'B', 'C')***

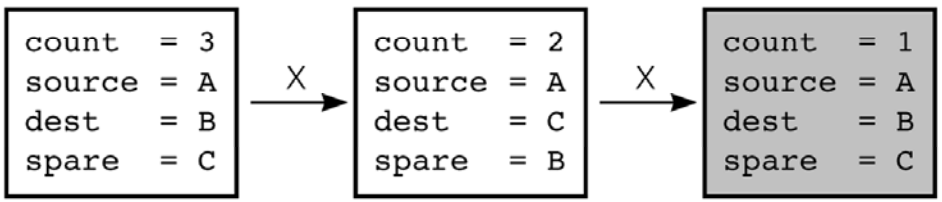
The initial call 1 is made, and **solveTowers** begins execution:



At point X, recursive call 2 is made, and the new invocation of the method begins execution:



At point X, recursive call 3 is made, and the new invocation of the method begins execution:



This is the base case, so a disk is moved, the return is made, and the method continues execution.

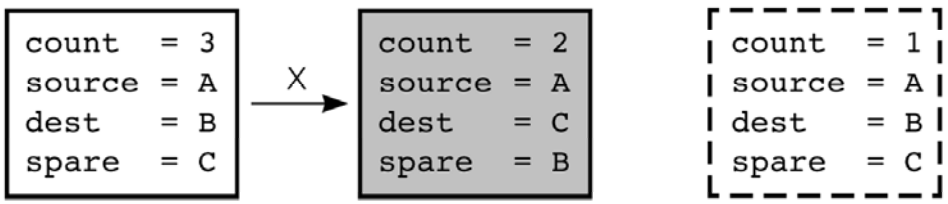
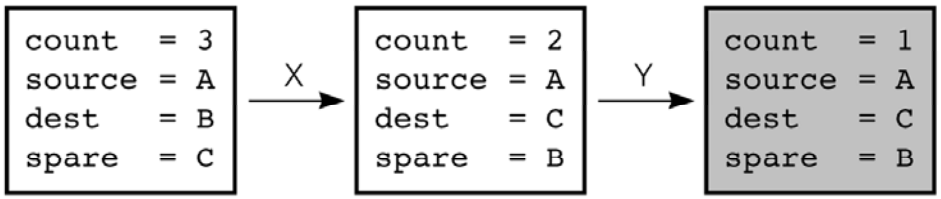


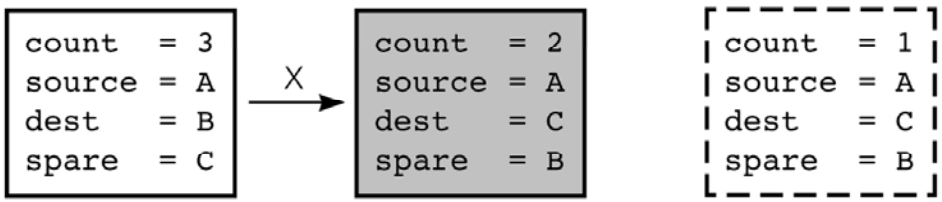
Figure 2.21b

Box trace of ***solveTowers(3, 'A', 'B', 'C')***

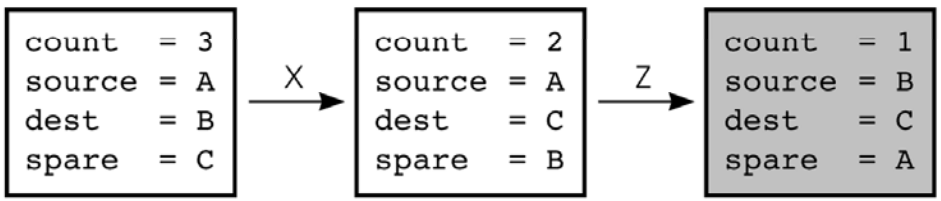
At point Y, recursive call 4 is made, and the new invocation of the method begins execution:



This is the base case, so a disk is moved, the return is made, and the method continues execution.



At point Z, recursive call 5 is made, and the new invocation of the method begins execution:



This is the base case, so a disk is moved, the return is made, and the method continues execution.

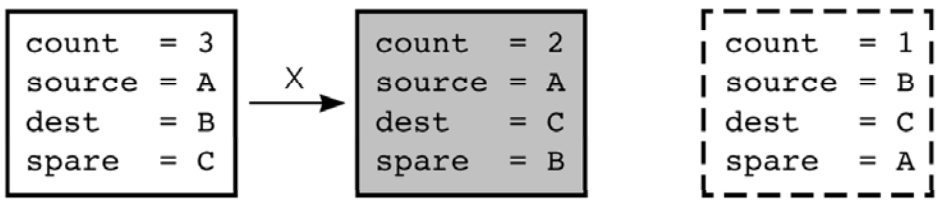
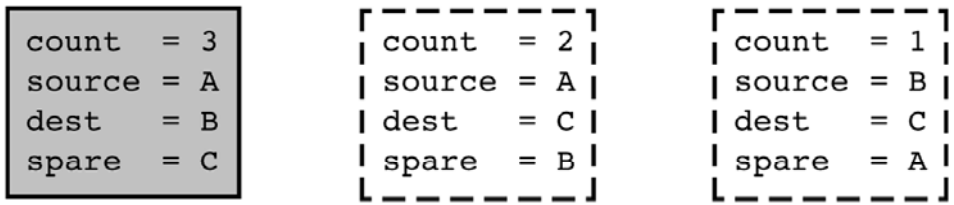


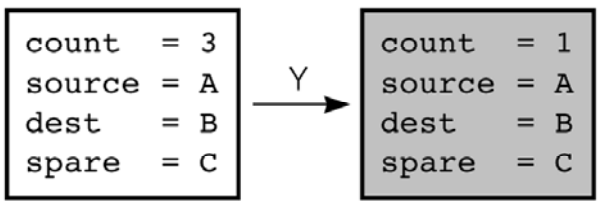
Figure 2.21c

Box trace of ***solveTowers(3, 'A', 'B', 'C')***

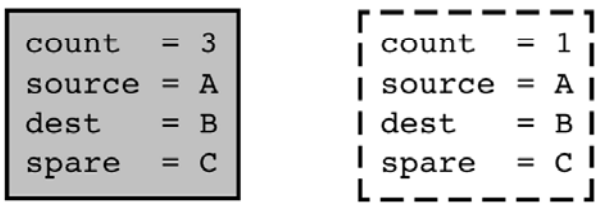
This invocation completes, the return is made, and the method continues execution.



At point Y, recursive call 6 is made, and the new invocation of the method begins execution:



This is the base case, so a disk is moved, the return is made, and the method continues execution.



At point Z, recursive call 7 is made, and the new invocation of the method begins execution:

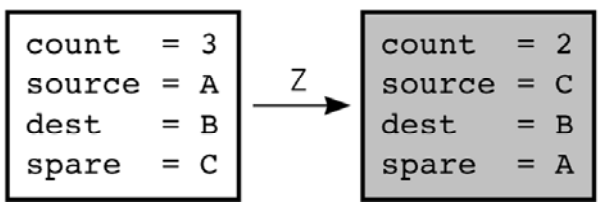
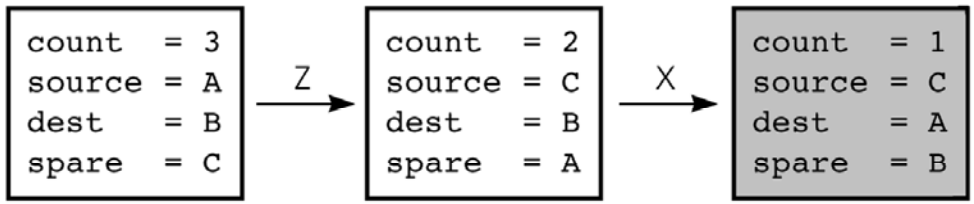


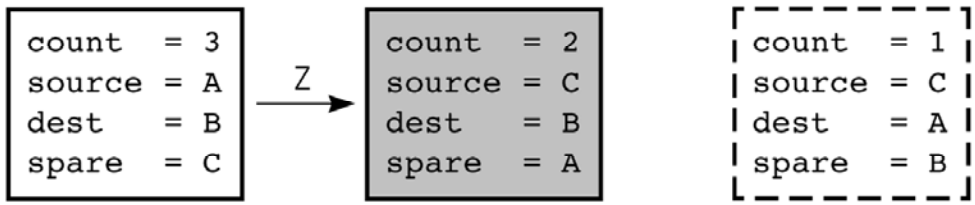
Figure 2.21d

Box trace of ***solveTowers(3, 'A', 'B', 'C')***

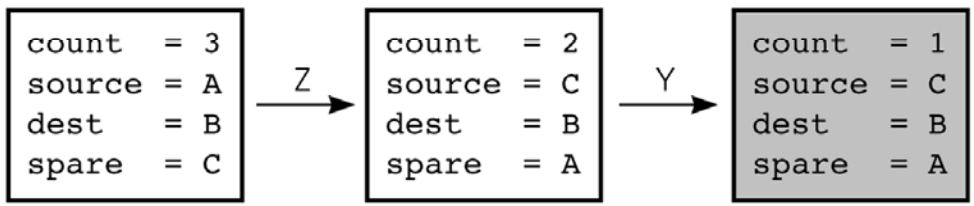
At point X, recursive call 8 is made, and the new invocation of the method begins execution:



This is the base case, so a disk is moved, the return is made, and the method continues execution.



At point Y, recursive call 9 is made, and the new invocation of the method begins execution:



This is the base case, so a disk is moved, the return is made, and the method continues execution.

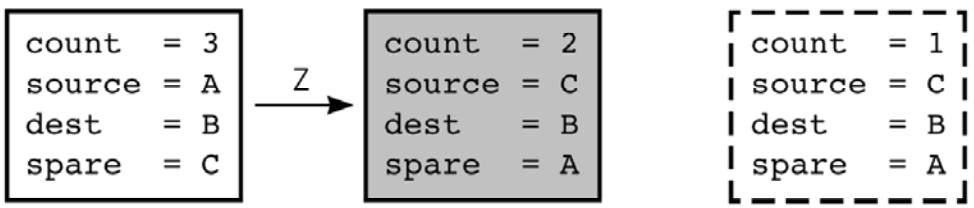
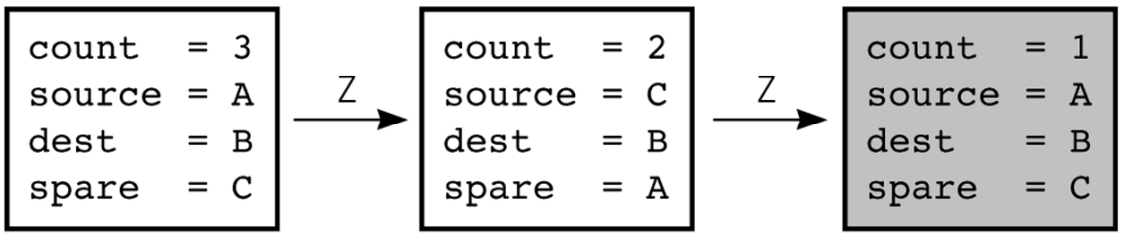


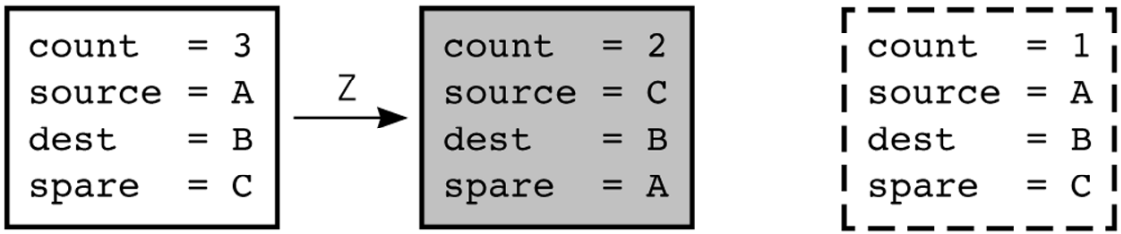
Figure 2.21e

Box trace of ***solveTowers(3, 'A', 'B', 'C')***

At point Z, recursive call 10 is made, and the new invocation of the method begins execution:



This is the base case, so a disk is moved, the return is made, and the method continues execution.



This invocation completes, the return is made, and the method continues execution.