# Solutions to the "travel" problem

First version: print all paths of length (x + y)

```
public static void travel(int x, int y) {
    travel(x + y, "moves:");
}

private static void travel(int stepsLeft, String path) {
    if (stepsLeft == 0) {
        System.out.println(path);
    } else {
        travel(stepsLeft - 1, path + " N");
        travel(stepsLeft - 1, path + " E");
        travel(stepsLeft - 1, path + " NE");
    }
}
```

Second version: prints final position as well

```
public static void travel(int x, int y) {
    travel(0, 0, x + y, "moves:");
}

private static void travel(int currX, int currY, int stepsLeft,
                            String path) {
    if (stepsLeft == 0) {
        System.out.println(path + ": (" + currX + ", " + currY + ")");
    } else {
        travel(currX, currY + 1, stepsLeft - 1, path + " N");
        travel(currX + 1, currY, stepsLeft - 1, path + " E");
        travel(currX + 1, currY + 1, stepsLeft - 1, path + " NE");
    }
}
```

Third version: prints only solutions

```
public static void travel(int x, int y) {
    travel(x, y, 0, 0, x + y, "moves:");
}

private static void travel(int currX, int currY, int x, int y,
                            int stepsLeft, String path) {
    if (currX == x && currY == y) {
        System.out.println(path);
    }
    if (stepsLeft > 0) {
        travel(x, y, currX, currY + 1, stepsLeft - 1, path + " N");
        travel(x, y, currX + 1, currY, stepsLeft - 1, path + " E");
        travel(x, y, currX + 1, currY + 1, stepsLeft - 1, path + " NE");
    }
}
```

Fourth version: stops searching after a solution is found

```
public static void travel(int x, int y) {
    travel(x, y, 0, 0, x + y, "moves:");
}

private static void travel(int currX, int currY, int x, int y,
                           int stepsLeft, String path) {
    if (currX == x && currY == y) {
        System.out.println(path);
    } else if (stepsLeft > 0) {
        travel(x, y, currX, currY + 1, stepsLeft - 1, path + " N");
        travel(x, y, currX + 1, currY, stepsLeft - 1, path + " E");
        travel(x, y, currX + 1, currY + 1, stepsLeft - 1, path + " NE");
    }
}
```

Fifth version: stops searching after a dead end is reached

```
public static void travel(int x, int y) {
    travel(x, y, 0, 0, "moves:");
}

private static void travel(int currX, int currY, int x, int y,
                           String path) {
    if (currX == x && currY == y) {
        System.out.println(path);
    } else if (currX <= x && currY <= y) {
        travel(x, y, currX, currY + 1, path + " N");
        travel(x, y, currX + 1, currY, path + " E");
        travel(x, y, currX + 1, currY + 1, path + " NE");
    }
}
```

Final version: keeps moves in a list structure, requires choose/explore/unchoose approach

```
public static void travel(int x, int y) {
    List<String> chosen = new ArrayList<String>();
    travel(x, y, 0, 0, chosen);
}

private static void travel(int currX, int currY, int x, int y,
                           List<String> chosen) {
    if (currX == x && currY == y) {
        System.out.println(chosen);
    } else if (currX <= x && currY <= y) {
        chosen.add("N");
        travel(x, y, currX, currY + 1, chosen);
        chosen.remove(chosen.size() - 1);

        chosen.add("E");
        travel(x, y, currX + 1, currY, chosen);
        chosen.remove(chosen.size() - 1);

        chosen.add("NE");
        travel(x, y, currX + 1, currY + 1, chosen);
        chosen.remove(chosen.size() - 1);
    }
}
```