Initial Example Counting Problem

Please read the following example, since the techniques and principles used through out it will be directly usable for the subsequent tutorial and example problems.

Billy has to travel to school every day. In order to actually get to school, Billy has to first get to the city. In order to get to the city, Billy can take:

- One of 2 buses
- One of 3 trains

Once Billy is at the city, he can then travel to his school. In order to get to his school, Billy can take:

- 1 bus
- One of 4 trains

Sometimes when Billy is running late or its raining, his mother will offer to drive him so that he avoids all of the public transport.

Using this information, Calculate the number of ways that Billy can get to School.

Answer

To get to the city, he can take either 2 different buses or 3 different trains

= 2 + 3 = 5 different ways to reach the city **(Rule of Sum)**

To get to his school from the city, he can take either 1 bus or 4 different trains

= 1 + 4 = 5 different ways to reach his school from the city **(Rule of Sum)**

Therefore, in order to get from his house to school he has a total of:

5 * 5 = 25 different ways **(Rule of Product)**

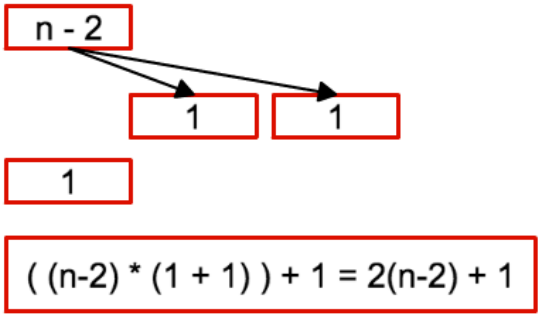
We must also add 1 for the times when his mother drives him:

= 25 + 1 = 26

Tutorial Problem

The solutions for these problems are similar to Big-O time-complexity in the sense that you always try to find the worst-case scenario.

```
public static int primeFactor(int n) {
    for (int divisor = 2; divisor < n; divisor++) {      n - 2
        if ((n % divisor) == 0) {      1
            return divisor;
        }      1
    }
    return 0;      1
}
```



((n-2) * (1 + 1)) + 1 = 2(n-2) + 1

With this problem, we start at the for-loop, which we can traverse n-2 times.
Within this loop we have an if-statement. We can either go in the if-statement or skip it (just like if it was an if-else statement) which results in 1 + 1 paths (similar to the counting example above, rule of sum).
Since we need to go through the loop n-2 times to reach the if-statement (rule of product), this result in:

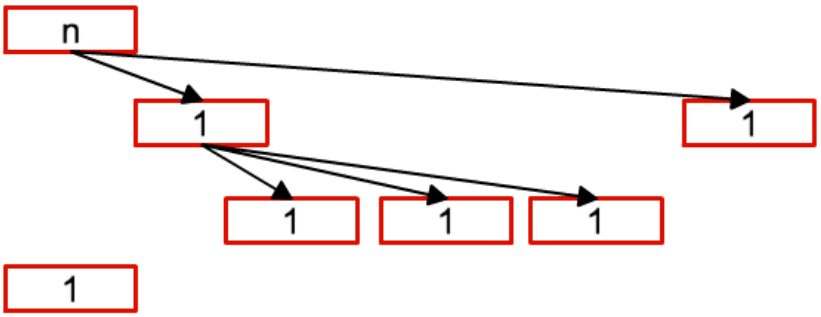
(n-2)(1 + 1) = 2(n-2)

Considering that there is a return statement within the if-statement, we might not reach the return statement outside of the for-loop. This means that we add an additional path to it which is one that reaches this return statement, so the final answer is:

2(n-2) + 1

Example Question 1

```
public static int evaluateNumber(int n, boolean mode) {
    for(int i = 0; i < n; i++) {          n
        if (mode == true) {               1
            if (n % 2) {                  1
                someFunction1();
            }
            else if (n % 3) {             1
                someFunction2();
            }
            else {                        1
                someFunction3();
            }
        }
        else {                            1
            someFunction4();
            return n;
        }
    }
    return 0;                             1
}
```



(n * ((1 * (1+1+1)) + 1) + 1
(n * ((1 * 3) + 1) + 1
(n * (3 + 1)) + 1 = 4n + 1

With this problem, we start at the for-loop, which we can traverse n times.
Within this loop we have an if-else statement:

1. The if-part contains an if-elseif-else statement. Considering that this results in three possible paths, it has a result of 3.
2. The else part does not contain any branching paths, so it only has a result of 1.

We add the total paths of the if-else statements to get a result of 3 + 1 = 4.

Since we need to go through the for-loop n times, this results in 4n, since the possible loop paths are executed all n times.

Considering that there is a return statement within the if-else-statement, we might not reach the return statement outside of the for-loop. This means that we need to add an additional path to it which is one that reaches the if-statement, so the final answer is:

$$4n + 1$$

Example Question 2

```
public static int operateArm(int n, int mode) {
    switch (mode) {
        case 0:
            someFunction1(); break;
        case 1:
            someFunction2(); break;
        case 2:
            someFunction3(); break;
        case 3:
            someFunction4(); break;
    }

    for (int i = 0; i < n; i++) {
        someFunction5();
    }

    if (someFunction6() && mode == true) {
        return n;
    }
    else {
        return 0;
    }
}
```

5

n

1

1

5 * n * 1 * 2 = 10n

With this problem we start at the switch-statement. Since there are 5 possible paths through this switch-statement (including the one where its skipped), this results in 5.
Next we have the for-loop which iterates n-times. However, since we could traverse through any of the 5 previous paths to get here, the result is 5n.
Finally, we have the if-else statement which has two possible paths through it. Considering we have taken 5n previous paths to get to this point, this results in 5n * 2 = 10n total paths.

Example Question 3

```
public static int processNumber(int n, boolean mode) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < 100; j++) {
            someFunction1();
        }
    }

    for (int i = 0; i < 20; i++) {
        if (mode == true) {
            someFunction2();
        }
        else {
            someFunction2();
        }
    }

    int k = someFunction3();
    switch(k) {
        case 0:
            someFunction4(); break;
        case 1:
            someFunction4(); break;
        default:
            break;
    }

    return n - k;
}
```

n

20

1 1

3

n * (20 * (1 + 1)) * 3

n * 40 * 3 = 120n

With this problem we start at the for-loop, which we traverse n-times. Inside the for-loop is another for-loop, however, this will run at a static 100 times, meaning that there aren't actually any branching paths for this as it will always run 100 times. Because of this, the result after the for-loop is only n.
Next is another for-loop which runs 20 times. Although this is static like the previous inner loop, inside it is a branching if-else statement. This means that, although the number of times that the for-loop runs is static, the path which could result for each of those times is different. Because of this, the result of this for-loop is 20 * (1 + 1) = 40. Once the first for-loop is taken into account, this leaves us with n * 40 = 40n.
Finally, we have a switch statement which has 3 possible paths through it (it wont be skipped since we have the default option inside it). Taking our previous paths into account, this results in 40n * 3 = 120n.

Note, the return statement at the end is not counted as a path, since regardless of what happens in the algorithm, this will always be hit.