211 Problems

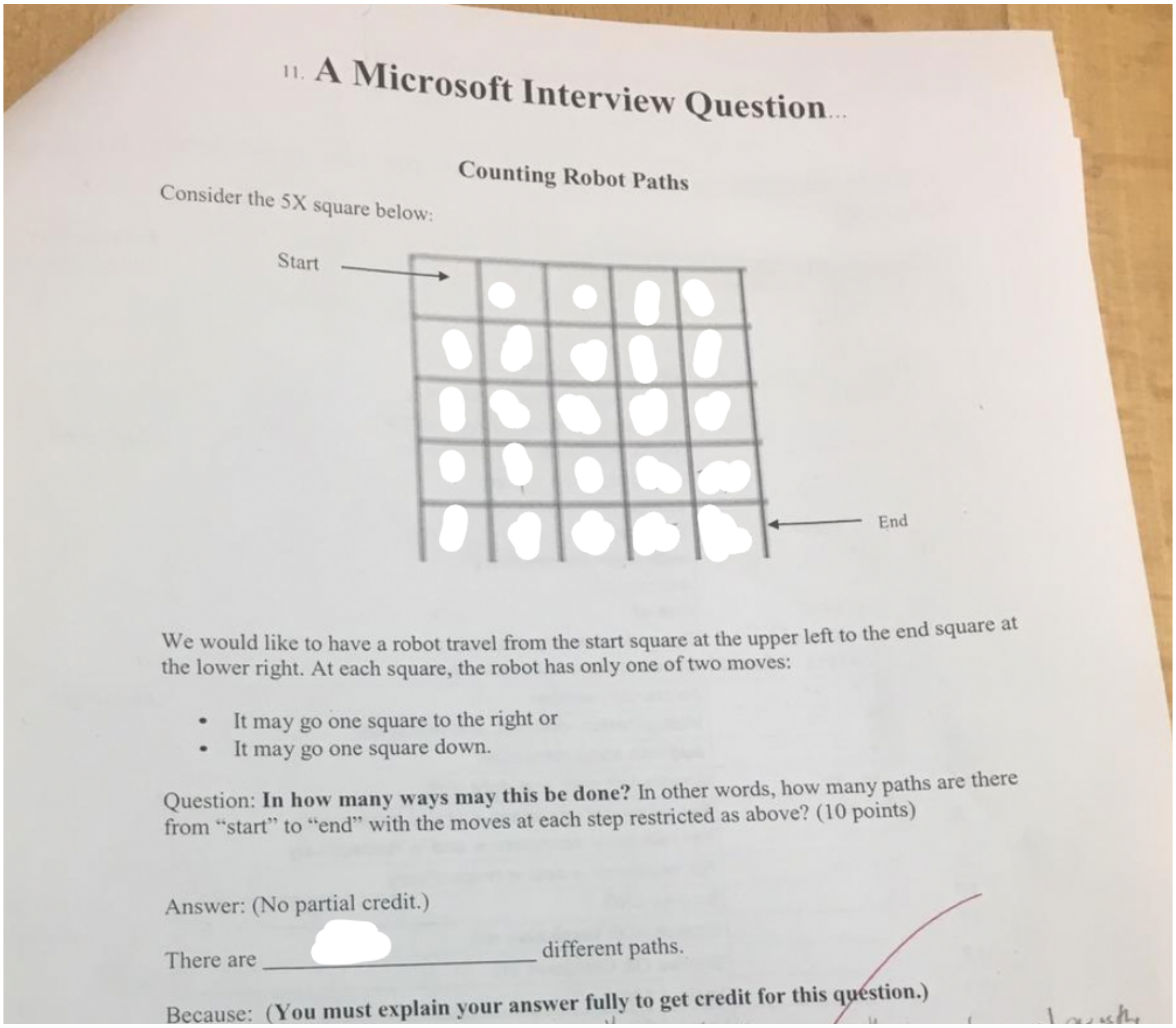
Robot Paths (Lattice Paths)

REMEMBER:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

A cool proof!

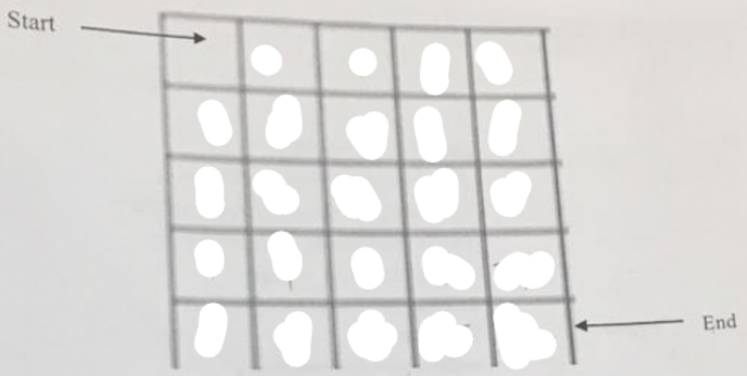
Here is the problem:



11. **A Microsoft Interview Question...**

Counting Robot Paths

Consider the 5X square below:



We would like to have a robot travel from the start square at the upper left to the end square at the lower right. At each square, the robot has only one of two moves:

- It may go one square to the right or
- It may go one square down.

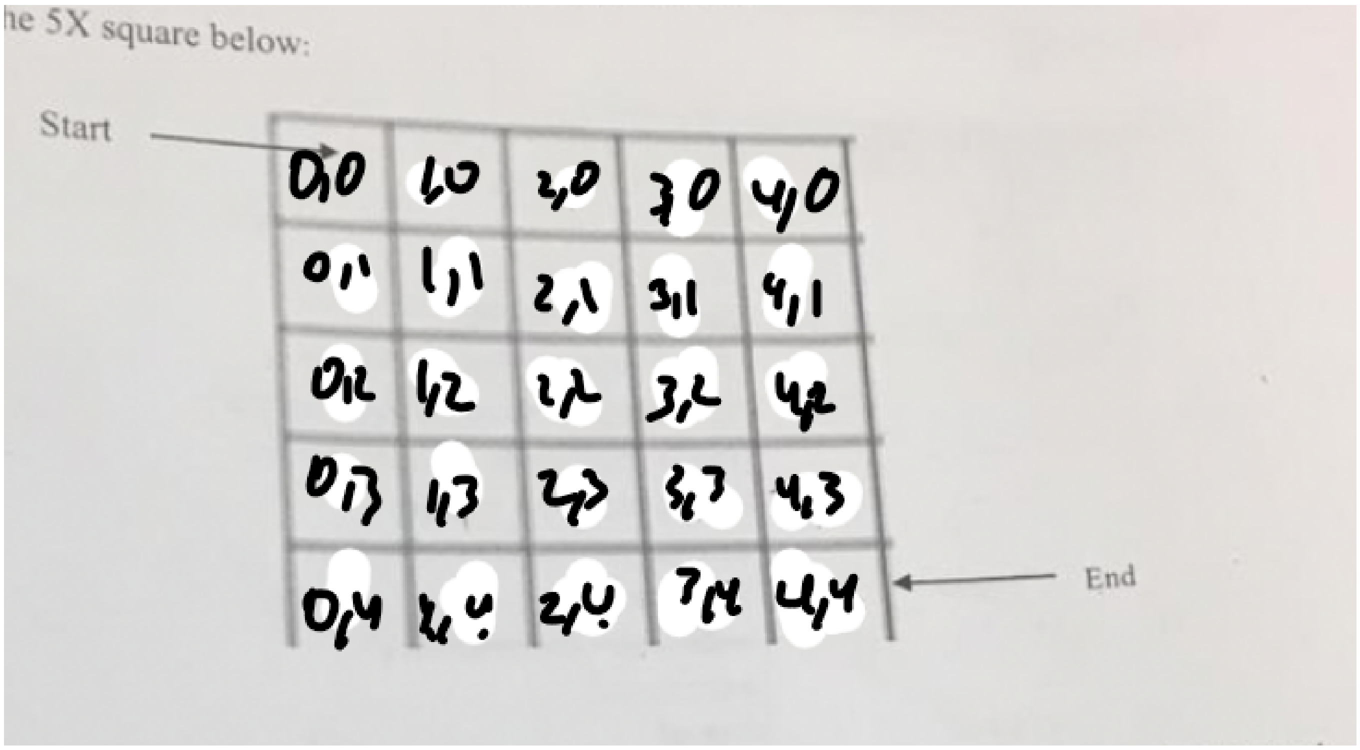
Question: **In how many ways may this be done?** In other words, how many paths are there from "start" to "end" with the moves at each step restricted as above? (10 points)

Answer: (No partial credit.)

There are [redacted] different paths.

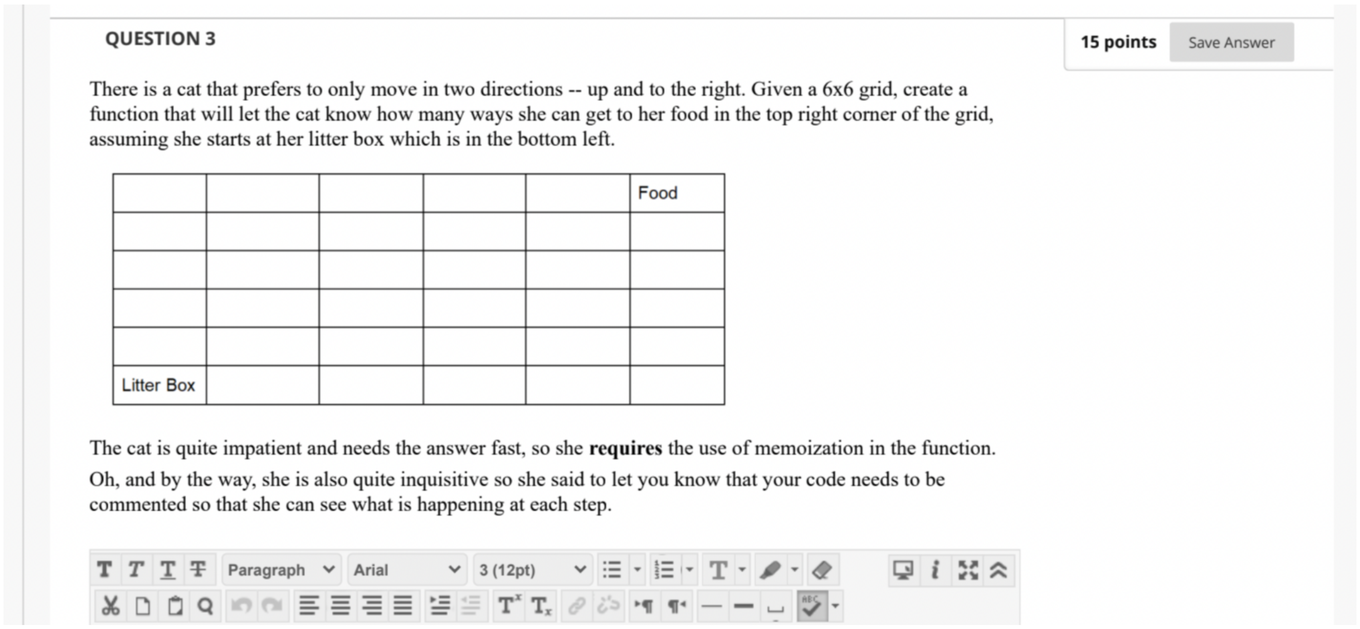
Because: **(You must explain your answer fully to get credit for this question.)**

This is similar to finding all the latice paths in a graph, so first we plot all of our points.



After plotting, we know that the # of lattice paths from point (0,0) to (x,y) which in this case is (4,4) will be equal to `x+y choose x`, in this case `4+4 choose 4` or `8 choose 4`. This translates to `8!/(4!(8-4!))`, solve for this and your result is `70`.

Path of Cats



QUESTION 3

15 points Save Answer

There is a cat that prefers to only move in two directions -- up and to the right. Given a 6x6 grid, create a function that will let the cat know how many ways she can get to her food in the top right corner of the grid, assuming she starts at her litter box which is in the bottom left.

					Food
Litter Box					

The cat is quite impatient and needs the answer fast, so she **requires** the use of memoization in the function. Oh, and by the way, she is also quite inquisitive so she said to let you know that your code needs to be commented so that she can see what is happening at each step.

We will be using the same method with lattice paths, and for that we need a way to calculate the factorials for the choose operation. This can be done using memoization so we don't have to re-calculate the same values, and we can save time.

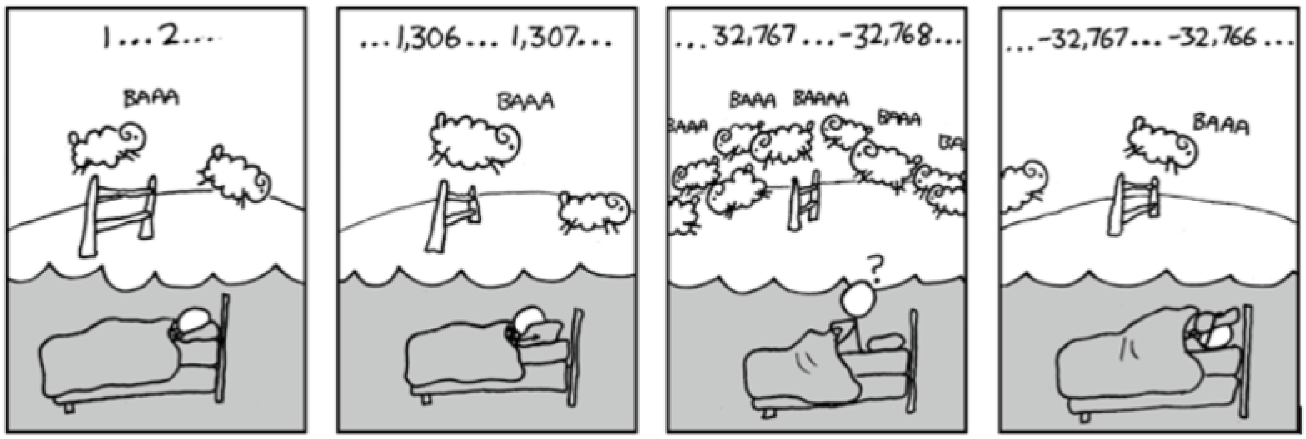
Below is the code:

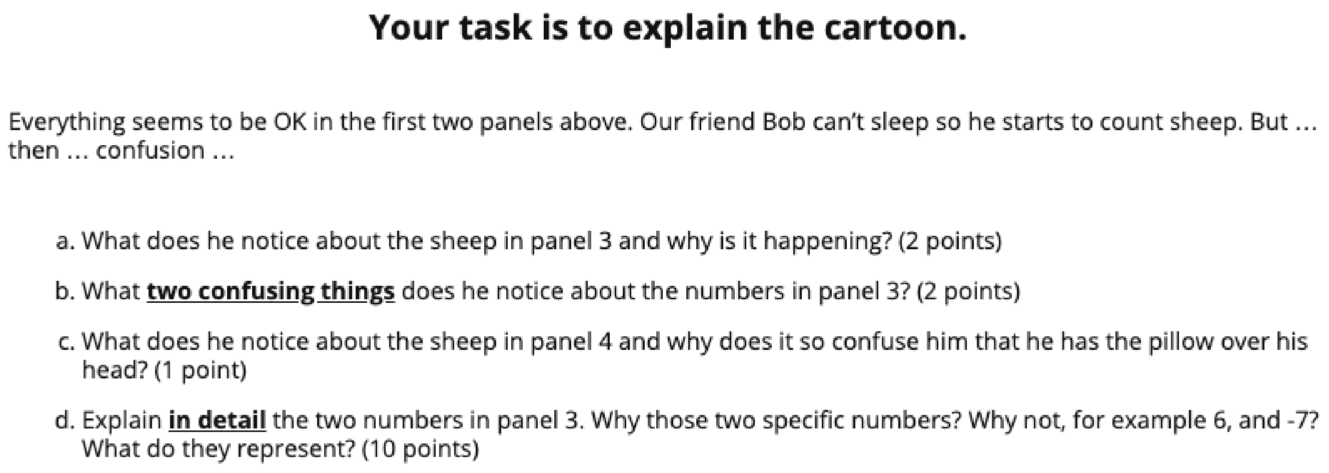
```
#include <iostream>
#include <vector>

using namespace std;

int factorial(int num, vector<int>& factorial_list){
    // If the number is 0 or 1, return 1 as their factorial is 1
    if(num == 1 || num == 0){
        return 1;
    }

    // If the requested factorial is already computed
    if(num <= factorial_list.size()){
        return factorial_list[num - 1];
    }

    // If the requested factorial is not computed, compute and store it
    for(int i = factorial_list.size(); i < num; i++){
        factorial_list.push_back(factorial_list.back() * (i + 1));
    }

    return factorial_list[num - 1];
}

int main()
{
    vector<int> factorial_list {1};  // Initialization of the factorial_list with
first element

    cout << "The total number of possible ways is: " << factorial(10,
factorial_list)/
                                                        (factorial(5,
factorial_list) * factorial(5, factorial_list)) << endl;

    return 0;
}
```

So we have the base case of 1, where 0! or 1! would return one. We then check if the factorial we are looking for is less than the size of the factorial_list, which starts from 1. If the number is less than the size, then that factorial is already stored and we can directly return it, otherwise, we will iterate from the latest stored factorial to the one we are searching for. This will be done iteratively. We would set the for loop to begin from the last element of the list (the latest computed factorial), then we would multiply the last computed factorial, `found through` `factorial_list.back()` by the last index + 1, this would result in the new factorial `+1` from the last index. This loop would continue until we have found and stored all of the values before the current factorial we are searching for. At the end of the loop, the last factorial value computed is returned.

Once we have this fuction ready, we can use the logic of finding the number of lattice paths. The # of lattice paths from point (0,0) to (x,y) which in this case is (5,5) will be equal to `x+y choose x` or 10 choose 5. This translates to `10!/(5!(10-5)!)`.

Sheep Problem





Your task is to explain the cartoon.

Everything seems to be OK in the first two panels above. Our friend Bob can't sleep so he starts to count sheep. But ... then ... confusion ...

a. What does he notice about the sheep in panel 3 and why is it happening? (2 points)

b. What **two confusing things** does he notice about the numbers in panel 3? (2 points)

c. What does he notice about the sheep in panel 4 and why does it so confuse him that he has the pillow over his head? (1 point)

d. Explain **in detail** the two numbers in panel 3. Why those two specific numbers? Why not, for example 6, and -7? What do they represent? (10 points)

a. He notices the numbers shift from positive to negative, this is happening because all of the possible sheep were counted, and now they are wrapping around to be counted from the beginning all over again.

b. The number turns negative, and then it counts one down.

c. The sheep are now running in the same way as when he was counting from 1 onwards, and it confuses him because negative sheeps are being counted, according to his counter.

d. The two numbers in panel 3 represent the range of numbers that can be stored in a 16 bit integer, this is -2^16 to 2^16 - 1. This is a result of two's complement where you use the first bit as sign bit.

Fibonacci Memoization

```
#include <iostream>
#include <vector>

using namespace std;

int fib(int num, vector<int> &fib_list){

    static int counter = 0;
    // If the number is 0 or 1, return 1 as their fib is 1
    if(num == 1 || num == 0){
        return 1;
    }

    // If the requested fib number is already computed
    if(num <= fib_list.size()){
        return fib_list[num - 1];
    }

    // If the requested fib is not computed, compute and store it
    for(int i = fib_list.size(); i < num; i++){
        counter++;
        cout << "Current count is:" << counter << endl;
        fib_list.push_back(fib_list.back() + fib_list[i - 2]);
    }

    return fib_list.back();
}

int main()
{
    vector<int> fib_list {1, 1};  // Initialization of the factorial_list with
first element

    cout << "Fib is:" << fib(4, fib_list) << endl;
    cout << "Fib is:" << fib(8, fib_list) << endl;
    return 0;
}
```

Factorial Memoization (Recursively)

```
#include <iostream>
#include <vector>

using namespace std;

vector <int> list{0}; // Initialize a vector with 100 zeros

int factorial(int num){
    if(num <= 1){
        return 1;
    }
    if(num < list.size()){
        return list[num];
    }

    list[num] = num * factorial(num - 1);
    return list[num];
}

int main()
{
    cout << "factorial: " << factorial(3);
    return 0;
}
```

Cannibals

```
struct State {
    int missionariesLeft;
    int cannibalsLeft;
    int missionariesRight;
    int cannibalsRight;
    bool boatLeft; // true if boat is on the left side, false if it's on the right
side

    bool isValidState() {
        if (missionariesLeft < 0 || cannibalsLeft < 0 || missionariesRight < 0 ||
cannibalsRight < 0) {
            return false;
        }
        if (missionariesLeft > 3 || cannibalsLeft > 3 || missionariesRight > 3 ||
cannibalsRight > 3) {
            return false;
        }
        if (missionariesLeft > 0 && missionariesLeft < cannibalsLeft) {
            return false;
        }
        if (missionariesRight > 0 && missionariesRight < cannibalsRight) {
            return false;
        }
        return true;
    }

    bool isGoalState() {
        return missionariesLeft == 0 && cannibalsLeft == 0 && missionariesRight ==
3 && cannibalsRight == 3;
    }

    bool operator==(const State& other) const {
        return missionariesLeft == other.missionariesLeft && cannibalsLeft ==
other.cannibalsLeft &&
               missionariesRight == other.missionariesRight && cannibalsRight ==
other.cannibalsRight &&
               boatLeft == other.boatLeft;
    }
};
```