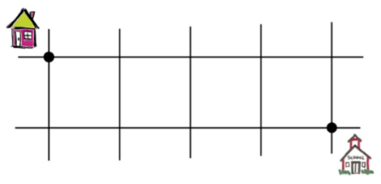
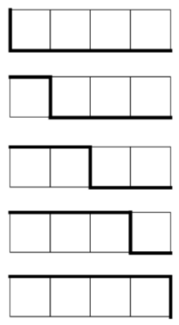
Assignment objectives

Practice the technique of *dynamic programming* and examine its time savings compared with straightforward recursion for applicable problems.

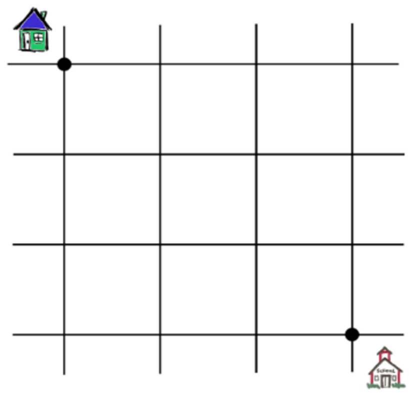
Introduction

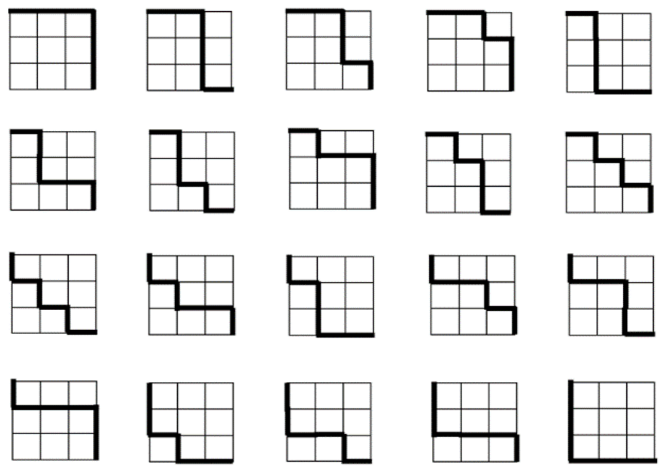
Young Johnny's daily walk to school is 5 blocks, as shown in the diagram below. He has a choice of five different routes he can follow:





Young Alice lives 6 blocks from the same school, but she can walk **20** different ways:





Question: Given any two such locations on a grid of city streets, how many different paths are there to walk from the first location to the second location? Assume that all walks proceed down and to the right.

We will calculate the answer to this question two ways:

- By a straightforward recursive method
- By a dynamic programming algorithm

Expressing the solution recursively

Define *SW(m, n)* to be the number of different ways to walk *m* blocks **down** and *n* blocks **over** on a grid of streets. There is no walking up or left. The starting point for a walk is location (0, 0).

Then *SW(1, 4)* = 5 is the number of ways Johnny can walk to school, and *SW(3, 3)* = 20 is the number of ways that Alice can walk to school.

SW(0, 0) is 1, because there is one and only one way to "go" to where you started: do nothing.

For any locations along the topmost (horizontal) street or the leftmost (vertical) street there is only 1 possible path: a straight line from the starting point. Therefore *SW(0, n)* and *SW(m, 0)* are both 1 for all values of m or n.

For calculating any other *SW(m, n)*, here is The Key Observation: To arrive at the point (m, n), you *must* first visit either the point immediately above, or the point immediately to the left. In either of these cases, there is only one way to complete your trip to (m, n).

Thus the total number of ways you can arrive to a point is equal to the SUM of the ways you could have arrived at those two neighboring points.

Therefore the function *SW(m, n)* can be expressed recursively as follows:

Base cases:

- If m = 0: *SW(0, n)* = 1 (walking *n* blocks straight to the right)
- If n = 0: *SW(m, 0)* = 1 (walking *m* blocks straight down)

Recursive case:

- If *m, n* are both > 0: *SW(m, n) = SW(m-1, n) + SW(m, n-1)*

Program design requirements

You must conform EXACTLY to all public-facing method signatures and identifier names, including capitalization and spelling!

Class name: Lab6

Public method: `long SW_Recursive(int, int)`
Arguments: the parameters of the SW() function

This method calculates and returns SW(m, n) by using the recursive definition given above, where m & n are the given parameters. Please note that the calculations of SW and the return value are of type `long`. **This function does NO console output.**

You can write a recursive helper method, or you can use SW_Recursive() itself recursively.

Public method: `void RunRecursive (int, int)`
Arguments: lower/upper bounds of a loop (***inclusive***)

Repeatedly calls SW_Recursive(), using arguments (i, i) for values of i that range over the bounds of the loop. Also uses one of the Java timing methods to measure the running time of each call to SW_Recursive. Produces console output that looks something like this (example showing output for calling RunRecursive(0, 5)):

```
SW_Recursive(0,0) = 1, time is 0 ms
SW_Recursive(1,1) = 2, time is 0 ms
SW_Recursive(2,2) = 6, time is 0 ms
SW_Recursive(3,3) = 20, time is 0 ms          ← Young Alice, 20 different walks!
SW_Recursive(4,4) = 70, time is 0 ms
SW_Recursive(5,5) = 252, time is 0 ms
```

You may use your own formatting and wording; the important thing is to display the required information (SW values and calculation time for each result).

NOTE: While you are testing SW_Recursive() and RunRecursive(), see if you can calculate values up to about SW(20, 20).

Public method: `long SW_DynamicProg(int, int)`
Arguments: the parameters of the SW() function

Calculates SW(m, n) by using dynamic programming. Please note that the calculations of SW and the return value are of type `long`. **This function does NO console output.**

Use an array to store values of SW(m, n) so that they can be looked-up/used in successive calculations. It's OK to use an ArrayList for this, but IMHO a plain array is the easiest, and I'm not just saying that because of my natural bias about arrays.

Public method: `void RunDynamicProg (int, int)`
Arguments: lower/upper bounds of a loop (***inclusive***)

Repeatedly calls SW_DynamicProg(), using arguments (i, i) for values of i that range over the bounds of the loop. Also uses one of the Java timing methods to measure the running time of each call to SW_DynamicProg. Produces console output that looks something like this (example showing output for RunDynamicProg(20, 24)):

```
SW_DynamicProg(20,20) = 137846528820, time is 0 ms
SW_DynamicProg(21,21) = 538257874440, time is 0 ms
SW_DynamicProg(22,22) = 2104098963720, time is 0 ms
SW_DynamicProg(23,23) = 8233430727600, time is 0 ms
SW_DynamicProg(24,24) = 32247603683100, time is 1 ms
```

You may use your own formatting and wording; the important thing is to display the required information (SW values and calculation time for each result).

NOTE: While you are testing SW_DynamicProg() and RunDynamicProg(), try calculating values up to at least SW(37, 37). (Note: The last few values will *fail*.)

Important Note

If you examine enough of these "SW" numbers (not just the (x, x) ones, but also the others), you may start to recognize them. You may also know a way to calculate them directly via a simple formula (involving factorials). *For the above required methods, you MUST NOT calculate the numbers this way.* The purpose of this assignment is not simply "calculate SW numbers"; it is really "write a program that uses/demonstrates Dynamic Programming".

Submission information

Due date: As shown on Learning Hub.

Submit the following to the drop box on Learning Hub:

- Just your Java source code (*.java file).
- File name is not important.
- Please *do not zip* or otherwise archive your code. Plain Java files only.
- Please *do not zip* or include your entire project directory.

Marking information

This lab is worth 20 points. As usual, 4-5 points are reserved for compliance with the COMP 3760 Coding Requirements.

Virtual donut 1

This part is not required; it is just for fun (and 🍩).

Ensure that your program can calculate the correct value of SW(37, 37), which is 1746130564335626209832. *Do not use/modify the two required SW functions for this bonus problem*; make a new function. Make it a public method so my test program can access it. Name it whatever you want. Write me a note with your submission so that I will know to look for it.

Once you're doing that, you can go quite a bit bigger. For example, this took my laptop about 2 seconds using the DP algorithm from this assignment:

SW(3737, 3737) = 7300230627418609431...083713739439587200 (2248 digits)

Virtual donut 2

This part is not required; it is just for fun (and 🍩).

Write code to compute the largest SW(x, x) that you can *by any algorithm*. This does not have to be Dynamic Programming! In fact, you can calculate SW numbers *waaaaaaay bigger and faster* with other techniques.

Submit the following information:

- The answer, in a text file of its own
- The number of digits
- The amount of computer time spent on the calculation (be sure to give the time units—seconds, minutes, hours?, days???)

Up to six people can earn VD2—one person in each set of COMP 3760. In addition, one person will receive *another* 🍩 for being the overall winner in all sets/both campuses combined. *I.e.*, just one person will earn 🍩🍩 for this bonus question!

Virtual donut 3

This part is not required; it is just for fun (and 🍩).

Write code that will compute SW(10000037, 10000037). That argument is *10 million and 37*.

Here are some results to help you know if your calculations are correct:

```
SW(1000037, 1000037) = 104463510588063...64189056000 (602080 digits)
SW(2000037, 2000037) = 724071786344346...11779937280 (1204139 digits)
SW(3000037, 3000037) = 579515833685420...08028979200 (1806199 digits)
SW(4000037, 4000037) = 491954088820849...46402475008 (2408259 digits)
SW(5000037, 5000037) = 431318349677477...56749056000 (3010319 digits)
SW(6000037, 6000037) = 385953997807883...92447436800 (3612379 digits)
SW(7000037, 7000037) = 350259806959580...04208640000 (4214439 digits)
SW(8000037, 8000037) = 321160730685731...57429693440 (4816499 digits)
SW(9000037, 9000037) = 296807076239095...01413606400 (5418559 digits)
SW(10000037, 10000037) = 276009199690603...72045537280 (6020619 digits)
```

Virtual donut 4

This part is not required; it is just for fun (and 🍩).

Beat my personal best. In December 2023 I calculated SW(666666666, 666666666), which has 401373323 digits. The calculation took about 63 minutes on my laptop, followed by another 83 minutes to convert it to a String and determine its length!

Needless to say, this should be done with your own code. Do not, for example, use Wolfram|Alpha, which is quite a bit faster than my code!