Fibonacci numbers

The **Fibonacci** sequence is named after Italian mathematician Leonardo of Pisa, known as Fibonacci:

https://en.wikipedia.org/wiki/Fibonacci_number

The **Fibonacci** numbers f_n = f(n) are the numbers characterized by the fact that every number after the first two is the sum of the two preceding ones. They are defined with the next recurrent relation:

$$f(n) = \begin{cases} 0, if \ n = 0 \\ 1, if \ n = 1 \\ f(n-1) + f(n-2) \end{cases}$$

So $f_0 = 0$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$.

The Fibonacci sequence has the form

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Example. Fill integer array *fib* with Fibonacci numbers (fib[i] = f_i):

```
#include <stdio.h>

int i, n, fib[47];

int main(void)
{
  scanf("%d",&n);

  fib[0] = 0; fib[1] = 1;
  for(i = 2; i <= n; i++)
    fib[i] = fib[i-1] + fib[i-2];

  printf("%d\n",fib[n]);
  return 0;
}
```

i	0	1	2	3	4	5	6	7	8	9	10	...
fib[i]	0	1	1	2	3	5	8	13	21	34	55	...

The biggest Fibonacci number that fits into `int` type is

f_{46} = 1836311903

The biggest Fibonacci number that fits into `long long` type is

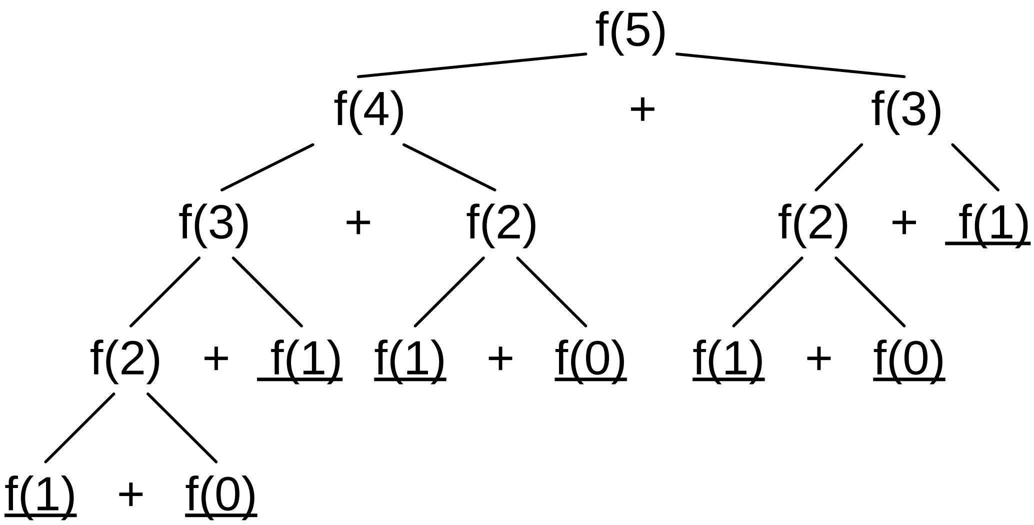
f_{92} = 7540113804746346429

If you want to find Fibonacci number f_n for n > 92, use **BigInteger** type.

Example. Find f(n) – the n-th Fibonacci number with recursion:

```
#include <stdio.h>
```

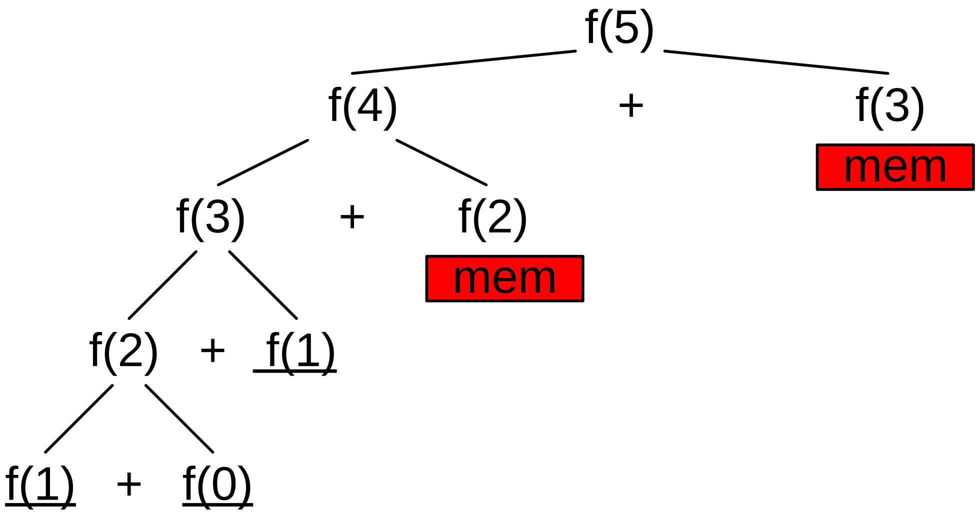
```
int n;

int fib(int n)
{
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib(n-1) + fib(n - 2);
}

int main(void)
{
  scanf("%d",&n);
  printf("%d\n",fib(n));
  return 0;
}
```



Example. Find f(*n*) – the *n*-th Fibonacci number with recursion + memorization:

```
#include <stdio.h>
#include <string.h>

int n, fib[46];

int f(int n)
{
  // base case
  if (n == 0) return 0;
  if (n == 1) return 1;

  // if the value fib[n] is ALREADY found, just return it
  if (fib[n] != -1) return fib[n];

  // if the value fib[n] is not found, calculate and memorize it
  return fib[n] = f(n-1) + f(n - 2);
}

int main(void)
{
  scanf("%d",&n);
```

```
  // fib[i] = -1 means that this value is not calculated yet
  memset(fib,-1,sizeof(fib));

  printf("%d\n",f(n));
  return 0;
}
```



Java code

```java
import java.util.*;

public class Main
{
  static int fib[] = new int[46];

  static int f(int n)
  {
    if (n == 0) return 0;
    if (n == 1) return 1;
    if (fib[n] != -1) return fib[n];
    return fib[n] = f(n-1) + f(n - 2);
  }

  public static void main(String[] args)
  {
    Scanner con = new Scanner(System.in);
    int n = con.nextInt();
    Arrays.fill(fib, -1);
    System.out.println(f(n));
    con.close();
  }
}
```

Prove the next **properties** for Fibonacci numbers:

a) $f_0 + f_1 + f_2 + f_3 + \ldots + f_n = f_{n+2} - 1$;

► Base case $n = 0$: $f_0 = f_2 - 1$, which is true because $0 = 1 - 1$.

Induction step: $f_0 + f_1 + f_2 + f_3 + \ldots + f_n = (f_0 + f_1 + f_2 + f_3 + \ldots + f_{n-1}) + f_n =$

$$f_{n+1} - 1 + f_n = f_{n+2} - 1$$

b) $f_1 + f_3 + f_5 + \ldots + f_{2n-1} = f_{2n}$;

► Base case $n = 1$: $f_1 = f_2$, which is true because 1 = 1.

Induction step: $f_1 + f_3 + \ldots + f_{2n+1} = (f_1 + f_3 + \ldots + f_{2n-1}) + f_{2n+1} =$

$$f_{2n} + f_{2n+1} = f_{2n+2}$$

c) $f_2 + f_4 + f_6 + \ldots + f_{2n} = f_{2n+1} - 1$;

► Base case $n = 1$: $f_2 = f_3 - 1$, which is true because 1 = 2 – 1.

Induction step: $f_2 + f_4 + \ldots + f_{2n+2} = (f_2 + f_4 + \ldots + f_{2n}) + f_{2n+2} =$

$$f_{2n+1} - 1 + f_{2n+2} = f_{2n+3} - 1$$

d) $f_0^2 + f_1^2 + f_2^2 + \ldots + f_n^2 = f_n * f_{n+1}$;

► Base case $n = 0$: $f_0^2 = f_0 * f_1$, which is true because 0 = 0 * 1.

Induction step: $f_0^2 + f_1^2 + f_2^2 + \ldots + f_n^2 = (f_0^2 + f_1^2 + f_2^2 + \ldots + f_{n-1}^2) + f_n^2 =$

$$f_{n-1} * f_n + f_n^2 = f_n * (f_{n-1} + f_n) = f_n * f_{n+1}$$

E-OLYMP [4730. Fibonacci] Fibonacci numbers is a sequence of numbers F(n), given by the formula:

$$F(0) = 1, F(1) = 1, F(n) = F(n - 1) + F(n - 2)$$

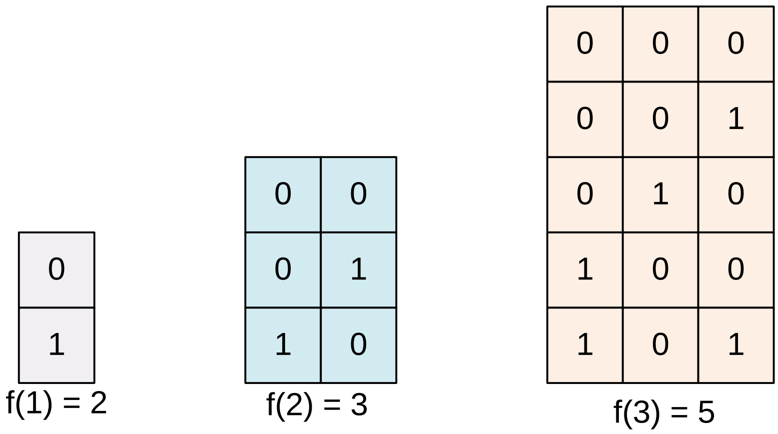
Given value of n ($n \le 45$). Find the n-th Fibonacci number.

► Implement a recursive function with memorization.

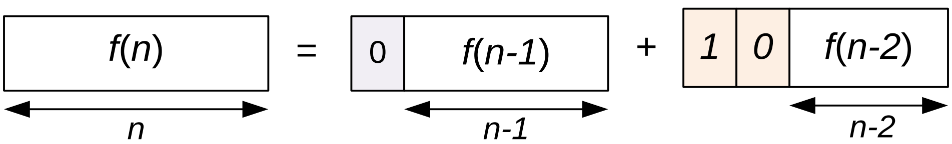
NO two one's in a row

Find the number of sequences of length n, consisting only of zeros and ones, that do not have two one's in a row.

Let $f(n)$ be the number of sequences consisting of 0 and 1 of length n that do not have two one's in a row.



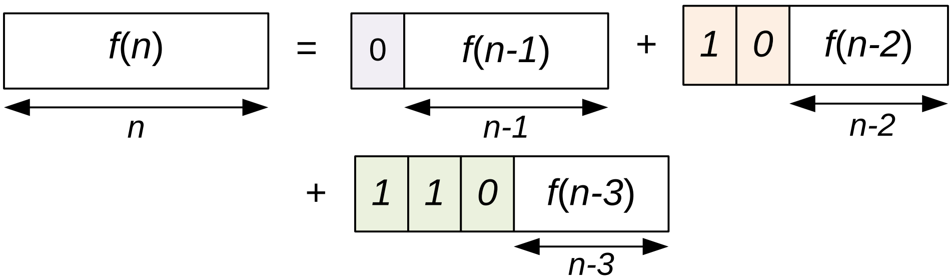
If the first number in the sequence is 0, then starting from the second place we can build $f(n - 1)$ sequences. If the first number in the sequence is 1, then second number should be 0.



We have Fibonacci numbers with base cases f(1) = 2, f(2) = 3.

E-OLYMP [263. Three ones](#) Find the number of sequences of length *n*, consisting only of zeros and ones, that do not have three one's in a row.

► Let *f*(*n*) be the number of required sequences consisting of 0 and 1 of length *n*. If the first number in the sequence is 0, then starting from the second place we can build *f*(*n* – 1) sequences. If the first number in the sequence is 1, then second number can be any (0 or 1). If second number is 0, on the next *n* – 2 free places we can construct *f*(*n* – 2) sequences. If second number is 1, the third number must be exactly 0, and starting from the forth place we can construct *f*(*n* – 3) sequences.



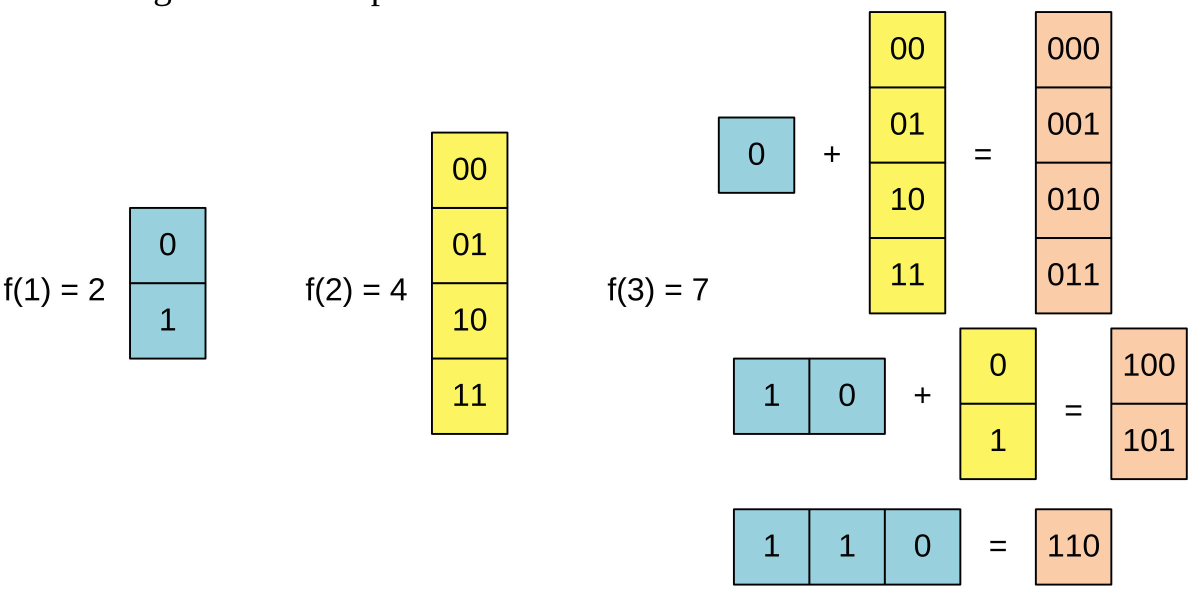
We have the recurrence: f(*n*) = f(*n* – 1) + f(*n* – 2) + f(*n* – 3). Now we must calculate the initial values:

f(1) = 2, since there are two sequence of lengths 1: 0 and 1.

f(2) = 4, since there are four sequence of lengths 2: 00, 01, 10 and 11.

f(3) = 7, since there are seven sequence of lengths 3: 000, 001, 010, 011, 100, 101 and 110.

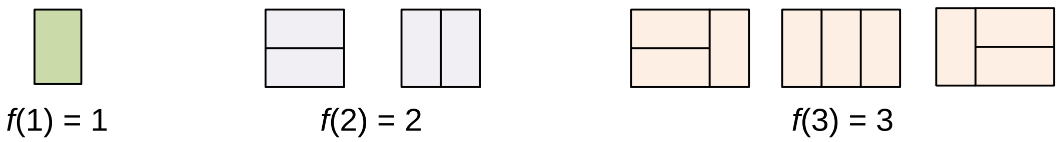
Do not forget to run all operations modulo 12345.



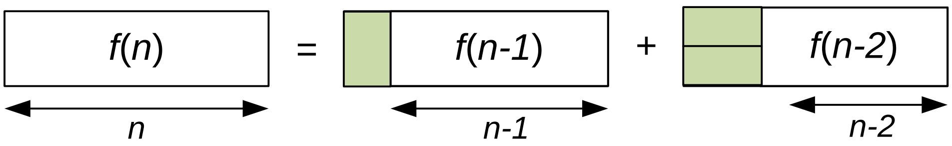
E-OLYMP [4469. Domino](#) Find the number of ways to cover a rectangle 2 × *n* with domino of size 2 × 1. The coverings that turn themselves into symmetries are considered different.

► Let f(*n*) be the number of ways to cover the 2 × *n* rectangle with 2 × 1 dominoes. Obviously, that

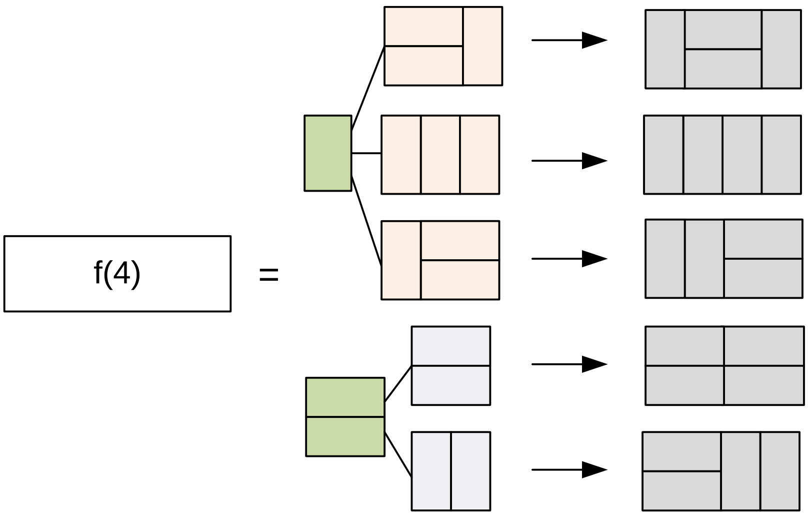
- *f*(1) = 1, one vertical domino;
- *f*(2) = 2, two vertical or two horizontal dominoes.



Consider an algorithm for computing $f(n)$. You can put one domino vertically and then cover a rectangle of length $n - 1$ in $f(n - 1)$ ways, or put two dominoes horizontally and then cover a rectangle of length $n - 2$ in $f(n - 2)$ ways. That is, $f(n) = f(n - 1) + f(n - 2)$.



So $f(n)$ is the Fibonacci number.



Since $n < 65536$, long arithmetic or Java programming language should be used.

E-OLYMP [8295. Fibonacci string generation](#) Generate the *n*-th Fibonacci string that is defined with the next recurrent formula:

- f(0) = "*a*";
- f(1) = "*b*";
- f(n) = f($n - 1$) + f($n - 2$), where "+" operation means concatenation

For example, f(3) = f(2) + f(1) = (f(1) + f(0)) + f(1) = "*b*" + "*a*" + "*b*" = "*bab*".

► Implement a recursive function that generates the *n*-th Fibonacci string.

```
string f(int n)
{
  if (n == 0) return "a";
  if (n == 1) return "b";
  return f(n-1) + f(n-2);
}
```

Read input value of *n* and print the *n*-th Fibonacci string.

```
cin >> n;
cout << f(n) << endl;
```

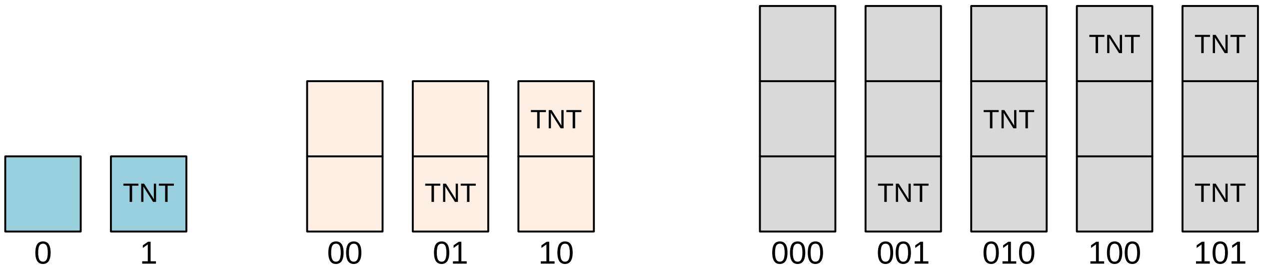
E-OLYMP 5091. Explosive containers You have two types of boxes: with trotyl (TNT) or without. You must build with boxes a tower of height n. In how many ways can you do it if it is forbidden to put TNT box on TNT box because of explosion.

► Let's code the empty box with 0 and the box with TNT with 1. In the problem we must find the number of strings of length n consisting of 0 and 1, in which two ones are not adjacent. The answer to the problem will be the Fibonacci number f(n):

$$f(n) = \begin{cases} 2, if \ n = 1 \\ 3, if \ n = 2 \\ f(n-1) + f(n-2) \end{cases}$$

Consider all possible towers of height n = 1, n = 2, n = 3. Each of them corresponds a sequence of 0 and 1. There are:

- two towers of height 1;
- three towers of height 2;
- five towers of height 3;

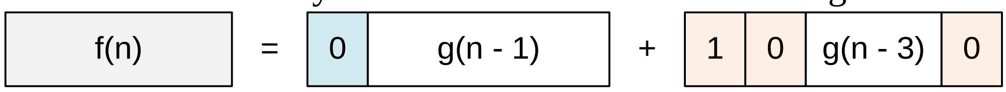


E-OLYMP 5103. Koza Nostra n teachers sit in a circle. The dealer should give to some of them one card with Ace (any amount of Aces is possible, even can be 0) – these teachers are the mafia. However, no two mafiosi can sit next to each other. In how many ways can deal the cards the dealer?

► Let g(n) be the number of ways to deal the cards to n teachers arranged in a row (the first teachers is not located next to the last). Then the problem is equivalent to finding the number of sequences of length n consisting of 0 and 1, where no two ones stand side by side. Solution to this problem is the Fibonacci number given by recurrence:

$$g(n) = \begin{cases} 2, if \ n = 1 \\ 3, if \ n = 2 \\ g(n-1) + g(n-2) \end{cases}$$

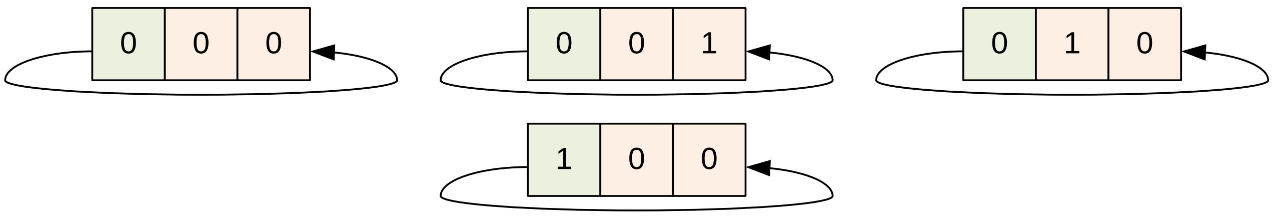
Let f(n) be the number of ways to deal cards to n teachers arranged in a circle.



If the first teacher was not given an ace, then the next n – 1 teachers can be given aces in g(n – 1) ways. If the first teacher was given an ace, then the second and last teachers should not be given aces. For the remaining n – 3 teachers the aces can be distributed in g(n – 3) ways. We have the relation:

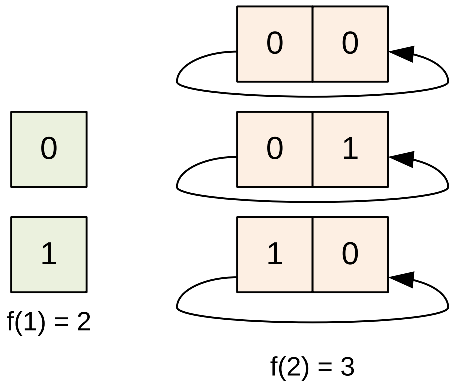
$$f(n) = g(n-1) + g(n-3), \text{ if } n \geq 3$$

For n = 3 we need the value g(0), that can be calculated from the equality g(0) + g(1) = g(2), whence g(0) = g(2) – g(1) = 3 – 2 = 1. Hence f(3) = g(2) + g(0) = 3 + 1 = 4.

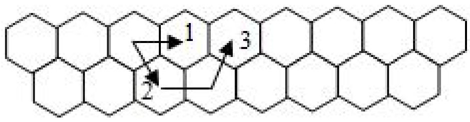


Here is the base cases:

- f(1) = 2
- f(2) = 3

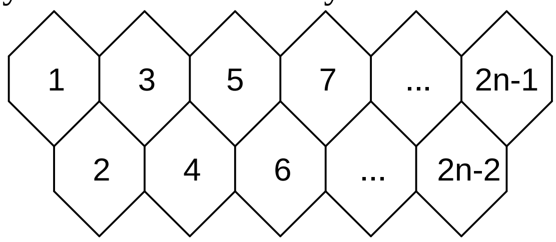


E-OLYMP [5092. Honeycomb](#) The bee can go in honeycomb as shown in the figure – with moves 1 and 2 from upper row and with move 3 from the lower.

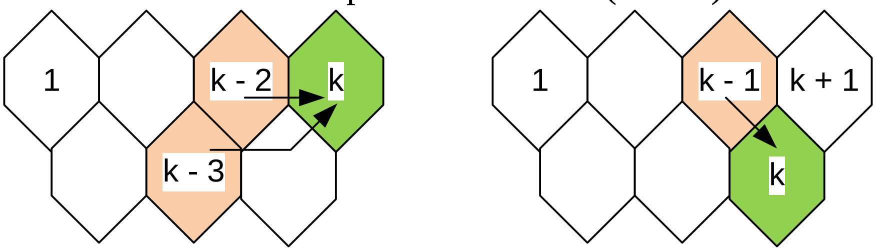


Find the number of ways to get from the first cell of the top row to the last cell of the same row.

► Enumerate the honeycomb in the next way:



Let f(k) be the number of ways to get from the first honeycomb into the k-th one. If upper row contains n honeycomb, the number of rightmost honeycomb of upper row has number 2n – 1. So the answer to the problem will be f(2n – 1).



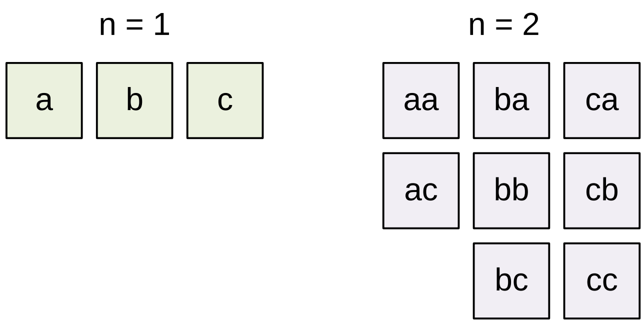
If k-th honeycomb is located in the upper row, the bee can come into it either from (k – 2)-th honeycomb, or from (k – 3)-th. So f(k) = f(k – 2) + f(k – 3) for odd k.

If k-th honeycomb is located in the lower row, the bee can come into it only from $(k-1)$-th honeycomb. So $f(k) = f(k-1)$ for even k.

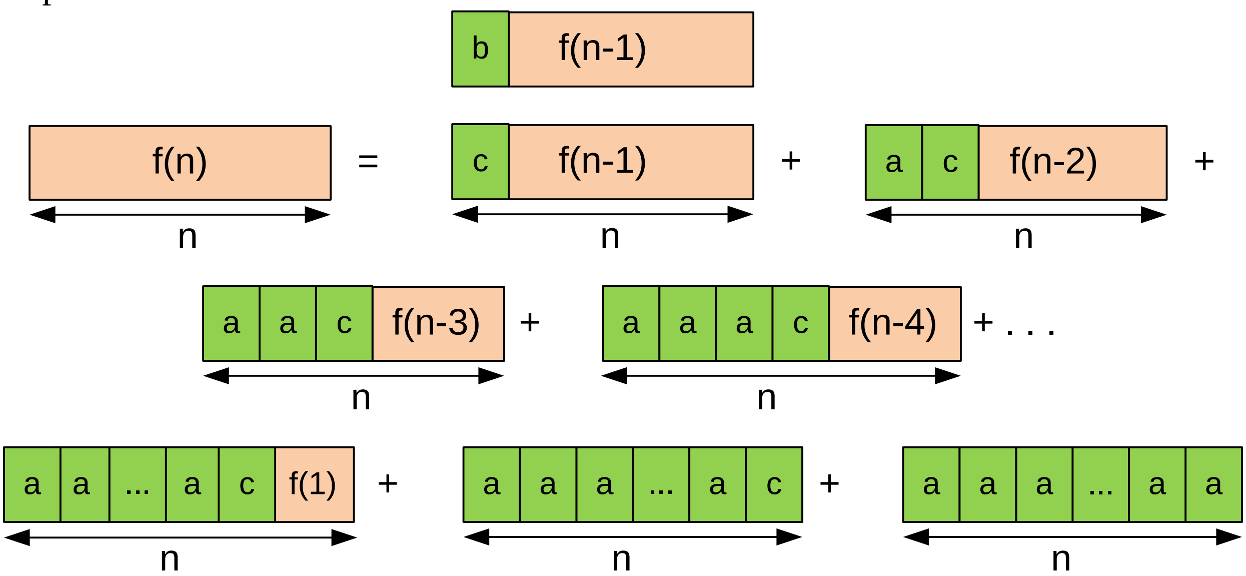
Calculate the base cases separately: $f(1) = 1$, $f(2) = 1$, $f(3) = 1$.

E-OLYMP 1343. Bad substring Find the number of strings of length n ($0 \le n \le 45$) consisting of only the characters '*a*', '*b*' and '*c*', not containing the substring "*ab*".

► Let $f(n)$ be the number of required strings of length n. If $n = 1$ we have 3 such strings, when $n = 2$ we have 8 strings:



Consider all possible ways to build the required strings. In the first position we can put one of three letters: '*a*', '*b*' or '*c*'. If we first put '*b*' or '*c*', then in the next $n-1$ positions we can put any of $f(n-1)$ words. If we first put '*a*', then we need to consider the cases of placing the letters in the second position. If we place in the second position '*c*', then in the next $n-2$ positions we can put any of $f(n-2)$ words. If we put in the second position '*a*', then similary we need to consider the placement of letters in the third position.



We have a relation:

$$f(n) = 2f(n-1) + f(n-2) + f(n-3) + \ldots + f(1) + f(0) + 1$$

How to simplify this recurrence? Let's rewrite it from $f(n-1)$:

$$f(n-1) = 2f(n-2) + f(n-3) + f(n-4) + \ldots + f(1) + f(0) + 1,$$

whence

$$f(n-2) + f(n-3) + f(n-4) + \ldots + f(1) + f(0) + 1 = f(n-1) - f(n-2)$$

Substitute this sum in the first relation:

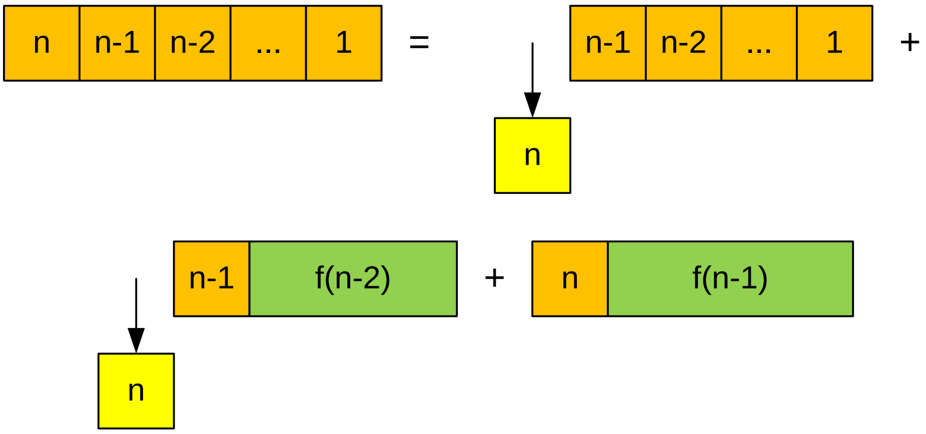
$$f(n) = 2f(n-1) + f(n-1) - f(n-2) = 3f(n-1) - f(n-2)$$

So we get the recurrence relation:

$$\begin{cases} f(n) = 3f(n-1) - f(n-2) \\ f(0) = 1, f(1) = 3 \end{cases}$$

E-OLYMP **5973. Out of the line!** *n* soldiers stay in one line. In how many ways can we choose some of them (at least one) so that among them there will not be soldiers standing in a line beside?

► Let f(*n*) be the number of ways for soldiers to out of the line. Its obvious that f(1) = 1 and f(2) = 2.



Let the soldiers in the row are numbered in decreasing order from *n* to 1. Then its possible to get out of the line with one of the next ways:

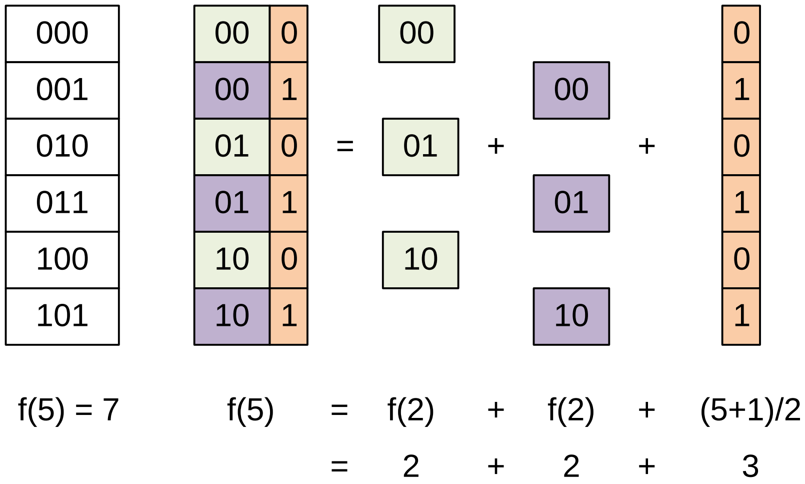
- *n*-th goes out, all others stay in a line;
- *n*-th goes out, then (*n* – 1)-st must stay in a line. Then recursively consider the solution for (*n* – 2) soldiers;
- *n*-th stay in a line. Then recursively solve the problem for (*n* – 1) soldiers;

So we get the recurrence relation:

$$\begin{cases} f(n) = f(n-1) + f(n-2) + 1 \\ f(1) = 1, f(2) = 2 \end{cases}$$

E-OLYMP **6583. Counting ones** How many ones in binary representation of numbers from 0 to *n*?

► Let f(*n*) be the number of ones in binary representation of all integers from 0 to *n*. Then the answer for the interval [*a*; *b*] is the value f(*b*) – f(*a* – 1).



If n is odd, then f(n) = 2 * f(n / 2) + $\lceil n/2 \rceil$.

If n is even, let f(n) = f(n – 1) + s(n), where s(n) is the number of ones in binary representation of n.

The base case is f(0) = 0.