Recursive functions

E-OLYMP [1207. Sqrt log sin](#) An evil professor has just assigned you the following problem. A sequence is defined by the following recurrence:

$$x_0 = 1,$$
$$x_i = x_{\lfloor i-\sqrt{i} \rfloor} + x_{\lfloor \ln(i) \rfloor} + x_{\lfloor i \sin^2(i) \rfloor}$$

Find $x_{1000000}$.

► The value of x_i will be calculated by means of the function f(i). To do it we must implement the recurrence

$$f(i) = f(\lfloor i-\sqrt{i} \rfloor) + f(\lfloor \ln(i) \rfloor) + f(\lfloor i\sin^2(i) \rfloor),$$
$$f(0) = 1$$

with memoization of f(i) in a linear array dp of size 10^6.

E-OLYMP [1211. Infinite sequence](#) Consider an infinite sequence A defined as follows:

$$A_0 = 1,$$
$$A_i = A_{[i/p]} + A_{[i/q]},\ i \ge 1$$

You will be given n, p and q. Find the value of A_n ($0 \le n \le 10^{12}$).

► To calculate all the values of sequence A_i (i = 0, 1, …, n) using an array is impossible because of the restriction $n \le 10^{12}$. To memoize the results we shall use a *map* structure: the value A_i will be stored in m[i]. Calculate the values of A_n memoizing the intermediate results.

To store the values of A_i declare the variable m.

```
map<long long,long long> m;
```

The function ***calc*** returns the value of m[n].

```
long long calc(long long n, int p, int q)
{
  if (n == 0) return 1;
  if (m[n] > 0) return m[n];
  return m[n] = calc(n/p,p,q) + calc(n/q,p,q);
}
```

E-OLYMP [1212. Infinite sequence - 2](#) Consider an infinite sequence A defined as follows:

$$A_i = 1,\ i \le 0,$$
$$A_i = A_{[i/p]-x} + A_{[i/q]-y},\ i \ge 1$$

You will be given n, p, q, x and y. Find the n-th ($0 \le n \le 10^{13}$) element of A.

► Since $n \le 10^{13}$, we can't store the values of the sequence A_i (i = 0, 1, …, n) neither by means of array, nor by means of *map* structure. Therefore, we implement the recursion as indicated in the recurrence relation, but at the same time the values of A_i, for which i < 5000000, will be stored in the array m.

To store the values of A_i (i < 5000000) declare the array m.

```
#define MAX 5000000
long long m[MAX];
```

The function ***calc*** computes the value of A_n.

```
long long calc(long long n, long long p, long long q,
               long long x, long long y)
{
  long long temp;
```

If $n \le 0$, then $A_n = 1$.

```
  if (n <= 0) return 1;
```

If n < 5000000 and the value m[n] is already computed (is not zero), return it.

```
  if ((n < MAX) && m[n] > 0) return m[n];
```

Perform a recursive calculation of the value A_n according to the formula given in the problem statement.

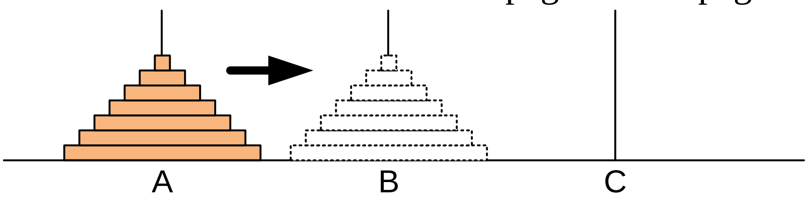
```
  temp = calc(n/p-x,p,q,x,y) + calc(n/q-y,p,q,x,y);
```

If n < 5000000, memoize A_n in m array to avoid recalculations.

```
  if (n < MAX) m[n] = temp;
  return temp;
}
```

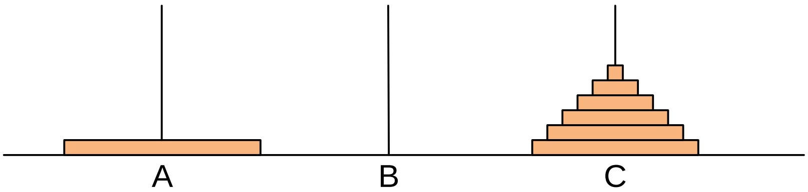
E-OLYMP [3936. Towers of Hanoi](#) Simulate the Hanoi Towers.

► Suppose we need to move n disks fron the peg A to the peg B using the peg C.

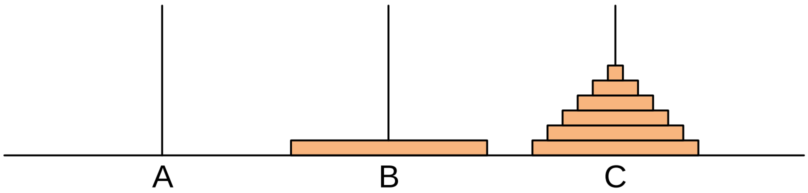


We shall use the following recursive scheme:

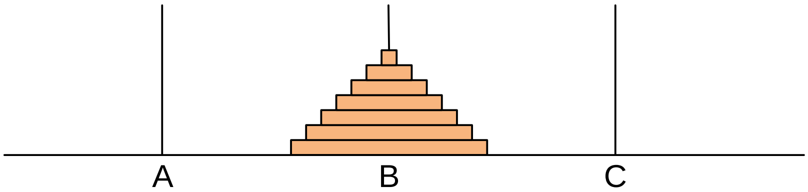
- move n – 1 disks from A to C using B;



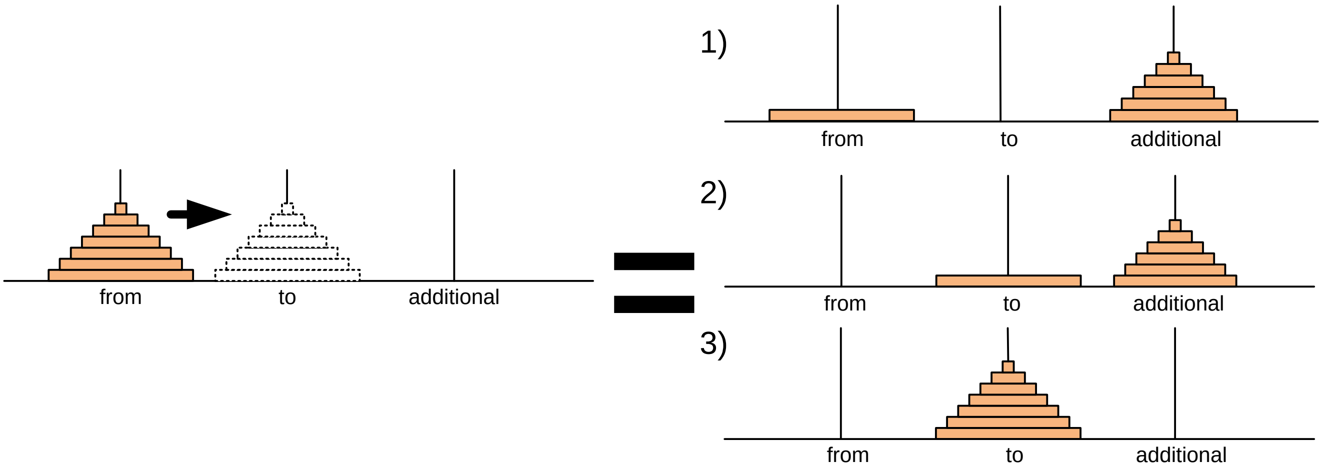
- move a disk from A to B;



- move n – 1 disks from C to B using A;



The function ***hanoi*** simulates the movement of disks from the peg *from* to the peg *to*, using an additionl peg *additional*.




```
void hanoi(int n, int from, int to, int additional)
{
  if (n == 0) return;
  hanoi(n-1,from,additional,to);
  printf("%d %d\n",from,to);
  hanoi(n-1,additional,to,from);
}
```

E-OLYMP [6155. Wrong monks] Find the minimum number of moves to solve the Hanoi Towers game with n discs.

► To move n disks from the first to the third peg, you should:

- move n – 1 discs from the first to the second peg
- move the remaining one (largest) disc from the first to the third peg
- move n – 1 discs from the second to the third peg

Let f(n) be the required number of moves. Then we get the recurrence relation:

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

Let's calculate some values of this function:

$$f(1) = 1,$$
$$f(2) = 2 * f(1) + 1 = 3,$$
$$f(3) = 2 * f(2) + 1 = 7,$$
$$f(4) = 2 * f(3) + 1 = 15$$

Let us prove by the method of mathematical induction that f(n) = $2^n - 1$.
Base step. f(1) = $2^1 - 1 = 1$.
Inductive step. f(n + 1) = 2 * f(n) + 1 = 2 * (2^n – 1) + 1 = $2^{n+1} - 1$.

Since $n \le 64$, the answer to problem $2^n - 1$ does not fit into a 64-bit signed type. Let's use an unsigned 64-bit integer type **unsigned long long**.

E-OLYMP 8304. Fun function Find the value of the function

$$f(x,y)=\begin{cases}0,\ x\le 0\ or\ y\le 0\\ f(x-1,y-2)+f(x-2,y-1)+2,\ x\le y\\ f(x-2,y-2)+1,\ x>y\end{cases}$$

It is known that $0 \le x, y \le 50$.

► Implement the recursive function with memoization. Declare two-dimensional array for storing the function values: dp[x][y] = f(x, y).

```
long long dp[51][51];
```

E-OLYMP 1520. Odd divisors Let f(n) be the greatest odd divisor of n, where n is a positive integer. You are given a positive integer n. Find the sum $f(1) + f(2) + ... + f(n)$.

► If n is odd, then $f(n) = n$. If n is even, then $f(n) = f(n / 2)$.

Let $g(n) = f(1) + f(2) + ... + f(n)$. Divide the set of positive integers from 1 to n into two subsets: odd ODD = {1, 3, 5, …, $2k$ – 1} and even EVEN = {2, 4, 6, …, $2l$} numbers.

ODD =	1	3	5	...	$2k$-1
EVEN =	2	4	6	...	$2l$

Among the positive integers from 1 to n there are exactly

$$k = \left\lfloor \frac{n+1}{2} \right\rfloor \text{ odd and } l = \left\lfloor \frac{n}{2} \right\rfloor \text{ even numbers}$$

Then $f(1) + f(3) + f(5) + ... + f(2k - 1) = 1 + 3 + 5 + … + (2k - 1) =$

$$\frac{1+2k-1}{2}\cdot k = k^2$$

At the same time $f(2) + f(4) + f(6) + ... + f(2l) = f(1) + f(2) + f(3) + ... + f(l) =$

$$g(l) = g\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

So $g(n) = k^2 + g\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$, where $k = \lfloor (n+1)/2 \rfloor$.

To stop the recursion we assume that g(0) = 0.

Consider the first test case, where n = 7.

Among the positive integers from 1 to 7 there are exactly $k = \lfloor (7+1)/2 \rfloor = 4$ odd and $l = \lfloor 7/2 \rfloor = 3$ even numbers.

ODD =	1	3	5	7
EVEN =	2	4	6	

$$g(7) = \left\lfloor \frac{7+1}{2} \right\rfloor^2 + g\left(\left\lfloor \frac{7}{2} \right\rfloor\right) = 16 + g(3) =$$

$$16 + \left\lfloor \frac{3+1}{2} \right\rfloor^2 + g\left(\left\lfloor \frac{3}{2} \right\rfloor\right) = 16 + 4 + g(1) = 16 + 4 + 1 = 21$$

g(7)	=	f(1)	+	f(2)	+	f(3)	+	f(4)	+	f(5)	+	f(6)	+	f(7)
	=	1	+	f(1)	+	3	+	f(2)	+	5	+	f(3)	+	7
	=	16	+	g(3)										

g(3)	=	f(1)	+	f(2)	+	f(3)
	=	1	+	f(1)	+	3
	=	4	+	g(1)		

E-OLYMP **1517. Simple addition** Let's define the next recursive function f(n), where

$$f(n) = \begin{cases} n\%10, \text{ if } n\%10 > 0 \\ 0, \text{ if } n = 0 \\ f(n/10) \text{ otherwise} \end{cases}$$

Let's define the function S(p, q) as follows:

$$S(p, q) = \sum_{i=p}^{q} f(i)$$

In this problem you have to calculate S(p, q) on given values of p and q.

► The function f(n) given in the problem statement finds the last non-zero digit of n. For example, f(1234) = 4, f(3900) = f(390) = f(39) = 9.

Let

$$g(p) = \sum_{i=1}^{p} f(i)$$

Then S(p, q) = g(q) – q(p – 1).

To compute the value of g(p), the sum of last significant digits for numbers from 1 to p, divide the numbers from 1 to p to three sets (‘/’ is an integer division):

1. Numbers from (p / 10) * 10 + 1 to p;
2. Numbers from 1 to (p / 10) * 10 that are not null-terminated;

3. Numbers from 1 to (p / 10) * 10 that are null-terminated;

The sum of the last significant digits in the first set equals to 1 + 2 + … + p mod 10 = t * (t + 1) / 2, where t = p mod 10. The sum of numbers in the second set equals to p / 10 * 45, because the sum of all digits from 1 to 9 equals to 45, and the number of full tens equals to p / 10. The required sum for the third set we shall find recursively: it equals to g(p / 10).We get a recurrence:

$$g(p) = \frac{t\cdot(t+1)}{2} + 45\cdot\left\lfloor\frac{p}{10}\right\rfloor + g\left(\left\lfloor\frac{p}{10}\right\rfloor\right),\ t = p \bmod 10$$

$$g(0) = 0$$

If p = 32, the fist set contains the numbers 31, 32, the second contains 1, …, 9, 11, …, 19, 21, …, 29, and the third contains 10, 20, 30. The value t = 32 mod 10 equals to 2.

I

31	32

$S_1 = 1 + 2 = 3$

II

1	2	3	...	9
11	12	13	...	19
21	22	23	...	29

$S_2 = 45 * 3 = 135$

III

10	20	30

1	2	3

$S_3 = f(1) + f(2) + f(3) = g(3)$

$$g(32) = \frac{2\cdot3}{2} + 45 * 5 + g(3) = 3 + 135 + (1 + 2 + 3) = 144$$

Let p = 1234.

I

1231	1232	1233	1234

$S_1 = 1 + 2 + 3 + 4 = 10$

II

1	2	3	...	9
11	12	13	...	19
. . .				
1221	1222	1223	...	1229

$S_2 = 45 * 123 = 5535$

III

10	20	30	...	1230

1	2	3	...	123

$S_3 = f(1) + f(2) + … + f(123) = g(123)$

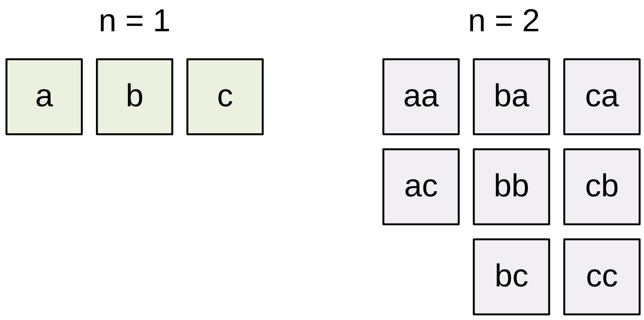
$$g(1234) = \frac{4\cdot5}{2} + 45 * 123 + g(123) = 10 + 5535 + g(123) = 5545 + g(123)$$

Computing the value g(123) = 595, we get:

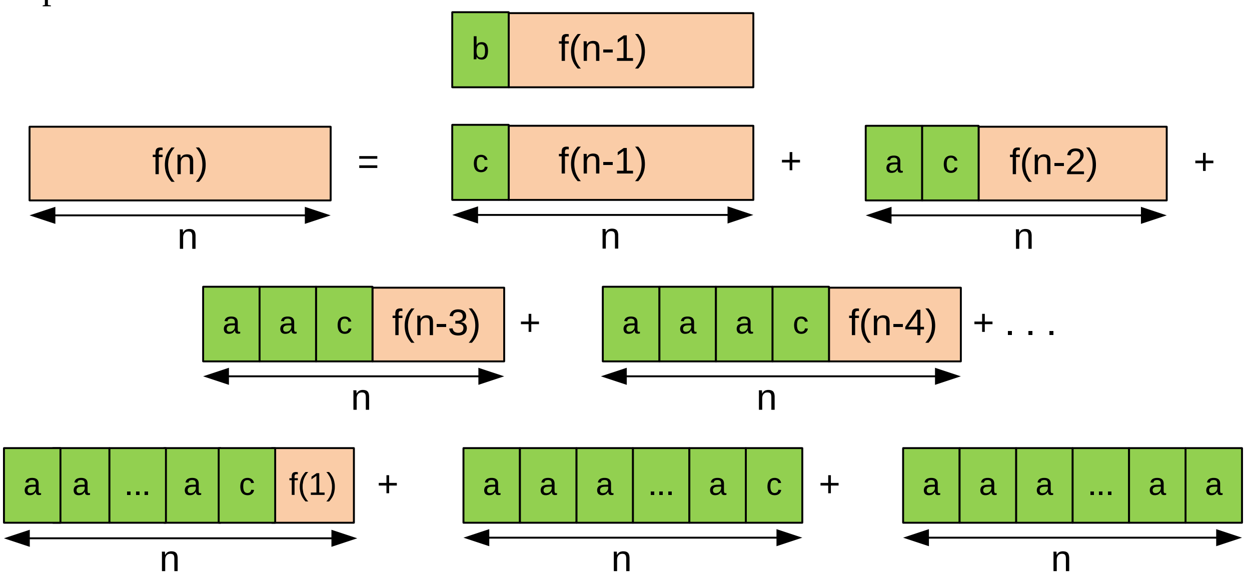
$$g(1234) = 5545 + g(123) = 5545 + 595 = 6140$$

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:

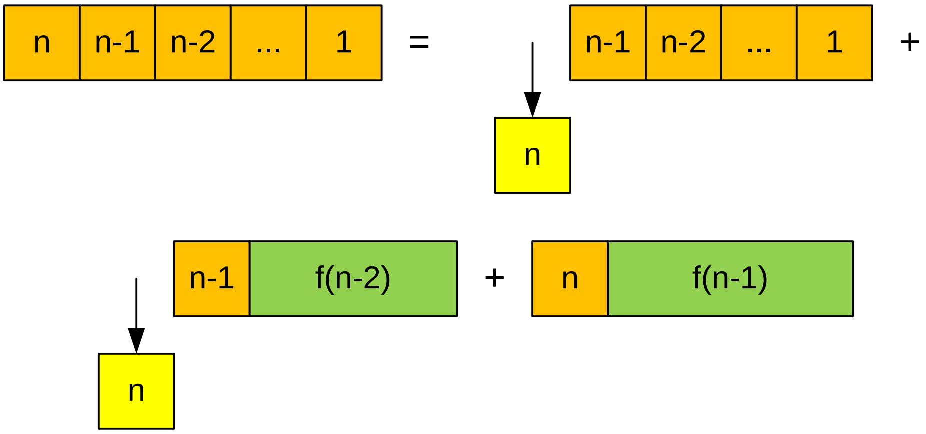
$$f(n) = 2f(n-1) + f(n-1) - f(n-2) = 3f(n-1) - f(n-2)$$

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:

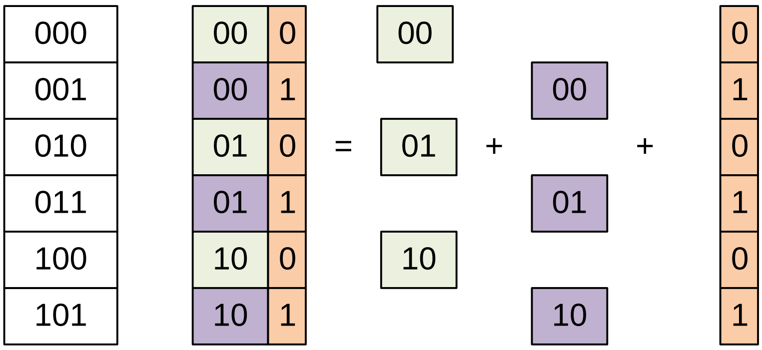
- *n*-th goes out, all others stay in a line;
- *n*-th goes out, then (*n* – 1)-st must stay in a line. Then recursively consider the solution for (*n* – 2) soldiers;
- *n*-th stay in a line. Then recursively solve the problem for (*n* – 1) soldiers;

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).



f(5) = 7 f(5) = f(2) + f(2) + (5+1)/2

= 2 + 2 + 3

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 $f(0) = 0$.

Consider the sample case, where $a = 2$, $b = 12$. The answer for the interval [2; 12] will be the value of $f(12) - f(1)$. There is one digit 1 in binary representation of the number 1, so $f(1) = 1$. Compute $f(12)$:

$f(12) = f(11) + s(12) = f(11) + 2$	**$f(12) = f(11) + 2 = 20 + 2 = 22$**
$f(11) = 2 * f(5) + \lceil 11/2 \rceil = 2 * f(5) + 6$	**$f(11) = 2 * f(5) + 6 = 2 * 7 + 6 = 20$**
$f(5) = 2 * f(2) + \lceil 5/2 \rceil = 2 * f(2) + 3$	**$f(5) = 2 * f(2) + 3 = 2 * 2 + 3 = 7$**
$f(2) = f(1) + s(2) = 1 + 1 = 2$	

The answer is $f(12) - f(1) = 22 - 1 = 21$.