ICC4101 - Algorithms and Competitive Programing

Javier Correa

Mathematics Problems

1. General/Adhoc problems:

- Simulation (brute force): simulate a mathematical process.
- Pattern/Formula: find a formula for the problem.
- Sequences: deal series of numbers such as Fibonnaci, or arithmetic/geometric series.
- Logarithm, exponentiation, power: clever uses of these functions.
- Polynomial, how to model, multiply, divide and derive polynomials.
- Base number variants, change between bases.
- Adhoc

2. Combinatorics

Problems related to counting different objects.

Binomial Coefficient $C(n,k)$

$C(n,k)$ is the number of wasy n elements can be taken k at a time.

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

Alternativelly:

- $C(n,0) = C(n,n) = 1$
- $C(n,k) = C(n-1,k-1) + C(n-1,k)$

implementing it with Dynamic Programing/Memoization.

Alternatively, `math.comb`.

Catalan numbers

Assuming that there are **two** types of steps (for example, *up* and *down*, or *right* and *left*), Catalan numbers count the number of way to perform $2n$ steps (for example moving from 0 to $2n$) such as in sequence the number of *up* steps is never less than the number of *down* steps.

Other examples:

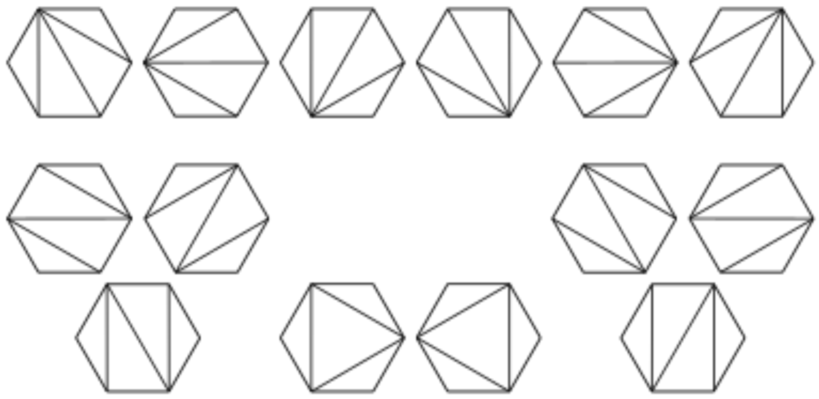
- The number of words of length $2n$ with characters 01 where no prefix has more 1's than 0's.

```
000111    010011    010101    001101    001011
```

- The number of correct parenthesizations of n parentheses or parenthesizations of $n + 1$ terms.

```
((ab)c)d    (a(bc))d    (ab)(cd)    a((bc)d)    a(b(cd))
```

- The number of ways that a convex polygon with $n + 2$ sides can be triangulated.



- The number of different full binary trees of n nodes.

Definition

$$Cat(n) = \frac{2n!}{n! \times (n+1)} = \frac{1}{n+1} C(2n, n)$$

These problems have satisfy the recurrence:

$$\begin{aligned} C(0) &= 1 \qquad (1)\\ C(n+1) &= \sum_{i=0}^{n} C(i)C(n-i) \qquad (2) \end{aligned}$$

Or alternativelly:

- $Cat(0) = 1$
- $Cat(n+1) = \frac{(2n+2)(2n+1)}{(n+2)(n+1)} Cat(n)$

3. Number theory

Having a basic knowledge about number theory marks the difference between a brute force solution and an elegant solution.

Prime numbers

Check if a number i is divisable by any number up to $\sqrt{i}$ (why?).

Prime factorization

Use the list of prime numbers to find primes factors of a given number. It is only required to check factors up to $\sqrt{N}$ (why?).

In [1]:
```
import math
from itertools import takewhile
def primes():
    i = 3
    prime_list=[2]
    yield 2
    while True:
        sqi = math.sqrt(i)
        if next((False for p in takewhile(lambda x: x <= sqi, prime_list) if
            prime_list.append(i)
            yield i
        i += 2
```

In [2]:
```
for i,p in enumerate(primes()):
    print(i, p)
    if i > 5:
        break
```

```
0 2
1 3
2 5
3 7
4 11
5 13
6 17
```

In [3]:
```
def factorize(N):
    """
    division: a / b          e.g. 2 / 1 = 2.0 : float
    integer division: a // b e.g. 2 // 1 = 2 : int
    remainder: a % b remainder of the integer division a / b = a // b + (a %
    4 % 2 = 0
    """
    sqN = math.sqrt(N)
    for p in primes(): # potentially infinite
        if p > sqN: # guarantees it ends
            yield N
            break
        while N % p == 0:
            yield p
            N //= p # N = N // p
            sqN = math.sqrt(N)
        if N == 1:
            break
```

In [4]:
```
list(factorize(23123123))
```

Out[4]:
```
[1607, 14389]
```

In [5]:
```
list(factorize(171617204849748568097))
```

Out[5]:
```
[11, 151, 51721, 1997672442637]
```

In [6]:
```
list(factorize(8))
```

Out[6]:
```
[2, 2, 2]
```

In [7]:
```
print(list(factorize(399069518807527340039904 3)))
```

```
[3, 23, 619, 2473, 18913, 1997672442637]
```

```
In [ ]: from itertools import islice

        class lazy_list:
            def __init__(self, gen):
                self.gen = gen
                self.mem = []

            def __iter__(self):
                class __iter:
                    def __init__(self, m):
                        self.i = 0
                        self.m = m
                    def __next__(self):
                        self.i += 1
                        return self.m[self.i-1]
                return __iter(self)

            def __getitem__(self, k):
                N = k - len(self.mem) + 1
                if N > 0:
                    self.mem += list(islice(self.gen, 0, N))
                return self.mem[k]
```

```
In [ ]: primes_ = lazy_list(primes())
```

```
In [ ]: primes_[5]
```

```
In [ ]: for i,p in enumerate(primes_):
            print(i,p)
            if i > 10:
                break
```

Other functions of interest

```
def prime_factor_count(N):
    """
    Count the number of prime factor of N. This is if
    N = p_1**a_1 * p_2**a_2 ... * p_k**a_k

    prime_factor_count(N) = a_1 + a_2 + ... + a_k
    """
    pass
```

```python
def different_prime_factor_count(N):
    """
    Count the number of unique prime factor of N. This is if
    N = p_1**a_1 * p_2**a_2 ... * p_k**a_k

    different_prime_factor_count(N) = k
    """
    pass

def number_of_divisors(N):
    """
    Number of different numbers that divide N This is if
    N = p_1**a_1 * p_2**a_2 ... * p_k**a_k

    number_of_divisors(N) = (a_1 + 1) * (a_2 + 1) ... * (a_k + 1)
    """
    pass

def euler_totient(N):
    """
    Number of relative primes of N
    N = p_1**a_1 * p_2**a_2 ... * p_k**a_k

    euler_totient(N) = p_1**(a_1 - 1) * (p_1 - 1) * p_2**(a_2 - 1)
* (p_2 - 1) ... p_k**(a_k - 1) * (p_k - 1)
    """
    pass
```

Greatest Common Divisor $gcd(a, b)$ and Least Common Multiple $lcm(a, b)$

```python
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)

def lcm(a, b):
    return a * (b / gcd(a, b))
```

Alternativelly, now the GCD is available as `math.gcd`.

Linear Diophantine Equation

Linear equation of integer coefficients and solution:

$$ax + by = \gcd(a, b)$$

where a, b, x, y must be integers.

Euclid algorithm can be extended to solve the following equation:

$$ax + by = \gcd(a, b) = d$$

This algorithm finds one solution x_1, y_1 with which all other possible solutions can be build using $x = x_1 + n\frac{b}{gcd(a,b)}, y = y_1 + n\frac{a}{gcd(a,b)}$ with n an integer.

```
def extended_euclid(a, b):
    if b == 0:
        return (1, 0, a)
    x, y, d = extended_euclid(b, a % b)
    return (y, x - (a // b) * y, d)
```

In [8]:
```
def extended_euclid(a, b):
    if b == 0:
        return (1, 0, a)
    x, y, d = extended_euclid(b, a % b)
    return (y, x - (a // b) * y, d)
```

Example Suppose a housewife buys apples and oranges with cost of $8.39. An apple is 25 cents. An orange is 18 cents. How many of each fruit does she buy?

The solution should follow this equation:

$$25x + 18y = 839$$

And, since you cannot buy fractional fruits, both x and y should be integers.

In [9]:
```
extended_euclid(25, 18)
```

Out[9]:
```
(-5, 7, 1)
```

Multipliying both sides by $\frac{839}{\gcd(25,18)}$ results in:

$$25\underbrace{(-5 \times 839)}_{x} + 18\underbrace{(7 \times 839)}_{y} = 839$$

Thus,

$$x = -4195 + 18n$$

$$y = 5873 - 25n$$

Since the number of apples and oranges must be positive, we also have the following constraints:

$$\begin{align}-4195 + 18n &> 0 \tag{3}\\ n &> \frac{4195}{18} \tag{4}\\ n &> 233.05 \tag{5}\end{align}$$

and

$$\begin{align}5873 - 25n &> 0 \tag{6}\\ \frac{5873}{25} &> n \tag{7}\\ 234.92 &> n \tag{8}\end{align}$$

Thus,the only solution is $n = 234$, which results in $x = 17$ apples and $y = 23$ oranges.

4. Modulo arithmetic

In some cases a problem can generate huge numbers, but we are asked for the number modulo N. This means that we are only interesed in the remainder of the result modulo N.

In `python` the modulo operator is given by `%`.

Properties of modular arithmetics

- If `a + b == c` then `(a % n) + (b % n) == (c % n)`.
- If `(a % n) == b` and `k` an integer then `(a + k) % n == (b + k) % n`.
- If `(a % n) == b` and `(c % n) == d` then `(a + c) % n == (b + d) % n`.

- If `a·b == c` then `(a % n)·(b % n) == c % n`.
- If `a % n == b` and `k` an integer then `a·k % n == b·k % n`.
- If `a % n == b` and `k` an integer then `a**k % n == b**k % n`.

In [ ]:
```python
a = [2, 1,2,3,4,5,65,6,8,8,9,6,5,3,3,4,45,5,6]
n = 7

from functools import reduce
print(sum(a) % n)
print(reduce(lambda x,y: (x+y) % n, a))
```

What is the difference between `sum(a) % n` and `reduce(lambda x,y: (x+y) % n, a)` ?

5. Probability

Two types of problems:

1. Direct formula:

$$P(a \wedge b) = P(a)P(b)$$

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}$$

- Count the total number of events and the number of positive events: $P(\text{event}) = \frac{\#\text{ set of positive events}}{\#\text{ complete set of event}}$

6. Game theoretic problems

These are problems related to two players playing a game and deciding if one of the players have a winning strategy given some initial conditions.

The general approach to solve these problems is to explore the different strategies via a Decision Tree or "Game Tree".

The starting node represent the initial state of the game and the current player. Each vertex is connected to a valid state of the game reachable from the current node but with a different playing player.

To explore this graph, is there is overlapping, Dynamic Programming is used otherwise standard backtracking is used.

At each node, the current player chooses the action with the larges winning margin (or the least loss).

In [ ]:
```
def game_tree(state, player):
    """ Pseudo code! """
    for a in actions(state):
        next_state = do_action(a, state)
        if winner(next_state) == player:
            return player
        if game_tree(next_state, next(player)) == player:
            return player
    return next(player)
```

Problem 1. Cola

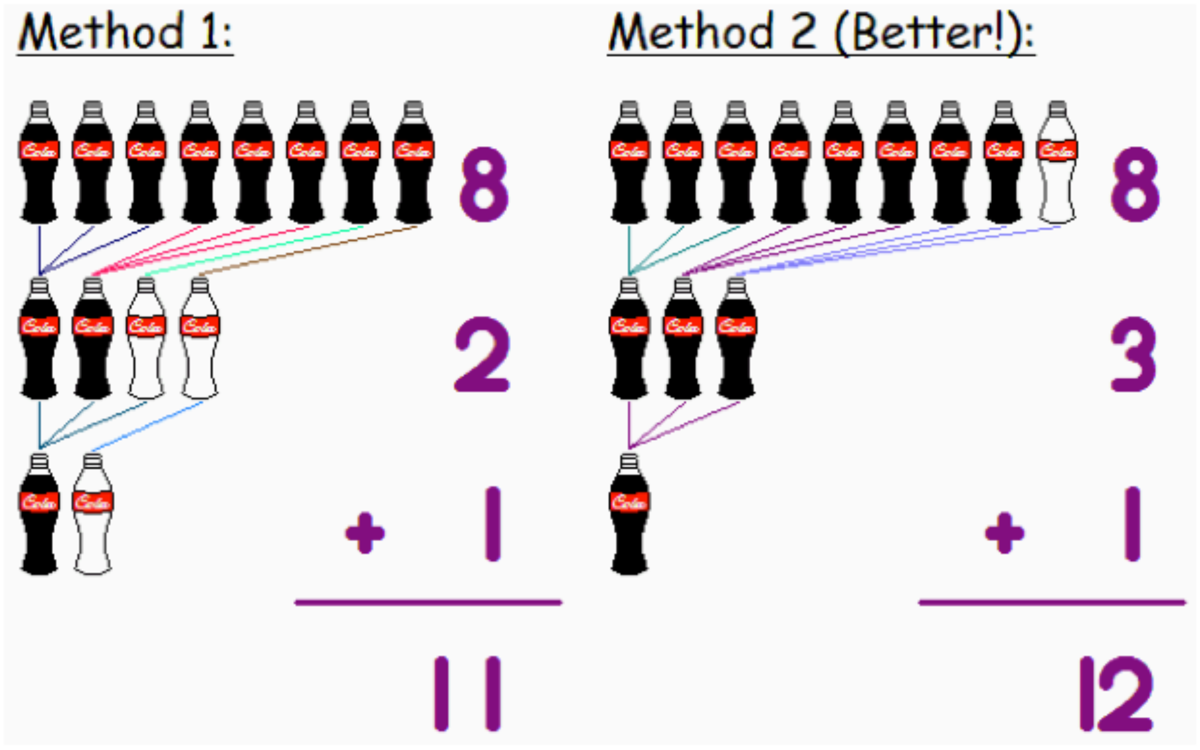
`https://www.udebug.com/UVa/11150`

You see the following special offer by the convenience store:

"*A bottle of Choco Cola for every 3 empty bottles returned*"

Now you decide to buy some (say N) bottles of cola from the store. You would like to know how you can get the most cola from them.

The figure below shows the case where N = 8. Method 1 is the standard way: after finishing your 8 bottles of cola, you have 8 empty bottles. Take 6 of them and you get 2 new bottles of cola. Now after drinking them you have 4 empty bottles, so you take 3 of them to get yet another new cola. Finally, you have only 2 bottles in hand, so you cannot get new cola any more. Hence, you have enjoyed 8 + 2 + 1 = 11 bottles of cola.



You can actually do better! In Method 2, you first borrow an empty bottle from your friend (?! Or the storekeeper??), then you can enjoy 8 + 3 + 1 = 12 bottles of cola! Of course, you will have to return your remaining empty bottle back to your friend.

Input

Input consists of several lines, each containing an integer N (1 ≤ N ≤ 200). Output For each case, your program should output the maximum number of bottles of cola you can enjoy. You may borrow empty bottles from others, but if you do that, make sure that you have enough bottles afterwards to return to them.

Note: Drinking too much cola is bad for your health, so... don’t try this at home!! :-)

Sample Input

```
8
```

Sample Output

```
12
```

Problem 2. The ? 1 ? 2 ? ... ? n = k problem

`https://www.udebug.com/UVa/10025`

Given the following formula, one can set operators ‘+’ or ‘-’ instead of each ‘?’, in order to obtain a given k

```
?1?2?...?n = k
```

For example: to obtain k = 12, the expression to be used will be:

```
- 1 + 2 + 3 + 4 + 5 + 6 - 7 = 12
```

with n=7

Input

The first line is the number of test cases, followed by a blank line. Each test case of the input contains an integer k (0 ≤ |k| ≤ 1000000000). Each test case will be separated by a single line.

Output

For each test case, your program should print the minimal possible n (1 ≤ n) to obtain k with the above formula.

Print a blank line between the outputs for two consecutive test cases.

Sample Input

```
2
12
-3646397
```

Sample Output

```
7
2701
```

Problem 3. Throwing Cards Away

`https://www.udebug.com/UVa/10940`

Given is an ordered deck of n cards numbered 1 to n with card 1 at the top and card n at the bottom. The following operation is performed as long as there are at least two cards in the deck: Throw away the top card and move the card that is now on the top of the deck to the bot- tom of the deck. Your task is to find the last, re- maining card.

Input

Each line of input (except the last) contains a positive number n ≤ 1000000. The last line contains '0' and this line should not be processed. Input will not contain more than 500000 lines.

Output

For each number from input produce one line of output giving the last remaining card.

Sample Input

```
7
19
10
6
0
```

Sample Output

```
6
6
4
4
```

Problem 4. Car

`https://www.udebug.com/UVa/11715`

You are in a car and going at the speed of u m/s. Your acceleration a is constant. After a particular time t, your speed is v m/s and your displacement is s. Now you are given some (not all of them) values for the given variables. And you have to find the missing parameters.

Input

The input file may contain multiple test cases. Each test case can be one of the

```
1 u v t
2 u v a
3 u a s
4 v a s
```

Input will be terminated by a single ‘0’.

Output

For each case of input you have to print one line containing the case number and

- If 1 u v t are given then print s and a
- If 2 u v a are given then print s and t
- If 3 u a s are given then print v and t
- If 4 v a s are given then print u and t

Check the samples for more details. You can assume that the given cases will not evaluate to an invalid situation. Use double for all calculations and output all floating point numbers to three decimal places.

Sample Input

```
1 10 5 2.0
1 5 10.0 2
2 10 11 2
3 5 1 6
4 5.0 -1 6
0
```

Sample Output

```
Case 1: 15.000 -2.500
Case 2: 15.000 2.500
Case 3: 5.250 0.500
Case 4: 6.083 1.083
Case 5: 6.083 1.083
```

Problem 5. How many nodes

`https://www.udebug.com/UVa/10223`

One of the most popular topic of Data Structures is Rooted Binary Tree. If you are given some nodes you can definitely able to make the maximum number of trees with them. But if you are given the maximum number of trees built upon a few nodes, Can you find out how many nodes built those trees?

Input

The input file will contain one or more test cases.

Each test case consists of an integer n (n ≤ 4,294,967,295). Here n is the maximum number of trees.

Output

For each test case, print one line containing the actual number of nodes.

Sample Input

```
5
14
42
```

Sample Output

```
3
4
5
```

Problem 6. Polly the Polynomial

`https://www.udebug.com/UVa/498`

Algebra! Remember algebra? There is a theory that as engineers progresses further and further in their studies, they lose basic math skills. This problem is designed to help you remember those basic algebra skills, make the world a better place, etc., etc.

Input

Your program should accept an even number of lines of text. Each pair of lines will represent one problem. The first line will contain a list of integers {c0, c1, . . . , cn} which represent a set of coefficients to a polynomial expression. The order of the polynomial is n. The coefficients should be paired with the terms of the polynomial in the following manner:

```
c0xn + c1xn−1 + ··· + cnx0
```

The second line of text represents a sequence of values for x, {x0 , x1 , . . . , xm }.

Output

For each pair of lines, your program should evaluate the polynomial for all the values of x (x0 through xm) and output the resulting values on a single line.

Sample Input

```
-2
5 0 1 6
1 -1
7 6 -1
```

Sample Output

```
-2
-2
-2
-2
6
5
-2
```

```
In [ ]:
```