```

###### ICPC用スニペット ######

from collections import defaultdict,deque
import sys,heapq,bisect,math,itertools,string,queue
sys.setrecursionlimit(10**8)
INF = float('inf')
mod = 10**9+7
eps = 10**-7
def inp(f): return int(f.readline())
def inps(f): return f.readline().rstrip()
def inpl(f): return list(map(int, f.readline().split()))
def inpls(f): return list(f.readline().split())

inpf = open('A.dat')
outf = open('Aout.dat',mode='w')

while True:
    x = inp(inpf)
    if x == 0:
        break

    outf.write('nya-n\n')



##### bit 関係 ######
#bit長
b = 10101
> b.bit_length() = 5

#一番最初のbitが立ってるものをとる
b&-b
> b    = 10110
> b&-b =    10

#ビットマスク(特定の桁だけ)
a = 10100
> (a>>0) & 1 = 0
> (a>>1) & 1 = 0
> (a>>2) & 1 = 1
> (a>>3) & 1 = 0
> (a>>4) & 1 = 1

#next_combination (n桁でk箇所bitが立ってる物を全探索)
def next_com(bit):
    x = bit & -bit
    y = bit + x
    return (((bit & ~y) // x) >> 1) | y
n,k = 5,3
bit = (1<<k)-1
ans = 0
while bit < (1<<n):
    for i in range(n):
        if (bit>>i) & 1:
            # 処理
    bit = next_com(bit)


#bitが立ってる数をカウント
def bitcount(bits):
    bits = (bits & 0x55555555) + (bits >> 1 & 0x55555555)
    bits = (bits & 0x33333333) + (bits >> 2 & 0x33333333)
    bits = (bits & 0x0f0f0f0f) + (bits >> 4 & 0x0f0f0f0f)
    bits = (bits & 0x00ff00ff) + (bits >> 8 & 0x00ff00ff)
    return (bits & 0x0000ffff) + (bits >>16 & 0x0000ffff)
```

```
##### itertools #####
seq = ('a', 'b', 'c', 'd', 'e')

# 並べ方
list(itertools.permutations(seq))
>[('a', 'b', 'c', 'd', 'e'),
  ('a', 'b', 'c', 'e', 'd'),
  ('a', 'b', 'd', 'c', 'e'),
  ('a', 'b', 'd', 'e', 'c'),
           中略
  ('e', 'd', 'c', 'a', 'b'),
  ('e', 'd', 'c', 'b', 'a')]

# 何個かを選ぶ並べ方
list(itertools.permutations(seq, 3))
> [('a', 'b', 'c'),
  ('a', 'b', 'd'),
  ('a', 'b', 'e'),
  ('a', 'c', 'b'),
       中略
  ('e', 'd', 'a'),
  ('e', 'd', 'b'),
  ('e', 'd', 'c')]

# 重複を許す順列 ([True,False]でやればbit全探索ができる)
list(itertools.product(A, repeat=3))
>[('a', 'a', 'a'),
  ('a', 'a', 'b'),
  ('a', 'a', 'c'),
  ('a', 'b', 'a'),
  ('a', 'b', 'b'), …

# 組み合わせ
list(itertools.combinations(seq,5))
> [('a', 'b', 'c', 'd', 'e')]

# 何個かを選ぶ組み合わせ
list(itertools.combinations(seq,3))
[('a', 'b', 'c'),
 ('a', 'b', 'd'),
 ('a', 'b', 'e'),
 ('a', 'c', 'd'),
 ('a', 'c', 'e'), …

# 重複を許す組み合わせ
list(itertools.combinations_with_replacement(A, 3))
[('a', 'a', 'a'),
 ('a', 'a', 'b'),
 ('a', 'a', 'c'),
 ('a', 'b', 'b'),
 ('a', 'b', 'c'), …


##### 編集距離 #####
S1 = 'yafo'
S2 = 'yahoo'
def levenshtein(s1,s2):
    n, m = len(s1), len(s2)
    dp = [[0]*(m+1) for _ in range(n + 1)]

    for i in range(n+1):dp[i][0] = i
    for j in range(m+1):dp[0][j] = j

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if s1[i-1] == s2[j-1]:cost = 0
            else:cost = 1
            dp[i][j] = min(dp[i - 1][j] + 1,         # insertion
                           dp[i][j - 1] + 1,         # deletion
                           dp[i - 1][j - 1] + cost)  # replacement
    return dp[n][m]
print(levenshtein(S1,S2))

```