

```

1
2 ##### ICPC用スニペット #####
3
4 from collections import defaultdict, deque
5 import sys, heapq, bisect, math, itertools, string, queue
6 sys.setrecursionlimit(10**8)
7 INF = float('inf')
8 mod = 10**9+7
9 eps = 10**-7
10 def inp(f): return int(f.readline())
11 def inps(f): return f.readline().rstrip()
12 def inpl(f): return list(map(int, f.readline().split()))
13 def inpls(f): return list(f.readline().split())
14
15 inpf = open('A.dat')
16 outf = open('Aout.dat', mode='w')
17
18 while True:
19     x = inp(inpf)
20     if x == 0:
21         break
22
23     outf.write('nya-n\n')
24
25
26
27 ##### bit 関係 #####
28 #bit長
29 b = 10101
30 > b.bit_length() = 5
31
32 #一番最初のbitが立ってるものをとる
33 b&-b
34 > b      = 10110
35 > b&-b =    10
36
37 #ビットマスク(特定の桁だけ)
38 a = 10100
39 > (a>>0) & 1 = 0
40 > (a>>1) & 1 = 0
41 > (a>>2) & 1 = 1
42 > (a>>3) & 1 = 0
43 > (a>>4) & 1 = 1
44
45 #next_combination (n桁でk箇所bitが立ってる物を全探索)
46 def next_com(bit):
47     x = bit & -bit
48     y = bit + x
49     return ((bit & ~y) // x) >> 1 | y
50 n,k = 5,3
51 bit = (1<<k)-1
52 ans = 0
53 while bit < (1<<n):
54     for i in range(n):
55         if (bit>>i) & 1:
56             # 処理
57     bit = next_com(bit)
58
59
60 #bitが立ってる数をカウント
61 def bitcount(bits):
62     bits = (bits & 0x55555555) + (bits >> 1 & 0x55555555)
63     bits = (bits & 0x33333333) + (bits >> 2 & 0x33333333)
64     bits = (bits & 0x0f0f0f0f) + (bits >> 4 & 0x0f0f0f0f)
65     bits = (bits & 0x00ff00ff) + (bits >> 8 & 0x00ff00ff)
66     return (bits & 0x0000ffff) + (bits >> 16 & 0x0000ffff)
67
68
69
70
71
72

```

```

73 ##### itertools #####
74 seq = ('a', 'b', 'c', 'd', 'e')
75
76 # 並べ方
77 list(itertools.permutations(seq))
78 >[('a', 'b', 'c', 'd', 'e'),
79   ('a', 'b', 'c', 'e', 'd'),
80   ('a', 'b', 'd', 'c', 'e'),
81   ('a', 'b', 'd', 'e', 'c'),
82   中略
83   ('e', 'd', 'c', 'a', 'b'),
84   ('e', 'd', 'c', 'b', 'a')]
85
86 # 何個かを選ぶ並べ方
87 list(itertools.permutations(seq, 3))
88 > [('a', 'b', 'c'),
89   ('a', 'b', 'd'),
90   ('a', 'b', 'e'),
91   ('a', 'c', 'b'),
92   中略
93   ('e', 'd', 'a'),
94   ('e', 'd', 'b'),
95   ('e', 'd', 'c')]
96
97 # 重複を許す順列 ([True,False]でやればbit全探索ができる)
98 list(itertools.product(A, repeat=3))
99 > [('a', 'a', 'a'),
100   ('a', 'a', 'b'),
101   ('a', 'a', 'c'),
102   ('a', 'b', 'a'),
103   ('a', 'b', 'b'), ...
104
105 # 組み合わせ
106 list(itertools.combinations(seq,5))
107 > [('a', 'b', 'c', 'd', 'e')]
108
109 # 何個かを選ぶ組み合わせ
110 list(itertools.combinations(seq,3))
111 [('a', 'b', 'c'),
112   ('a', 'b', 'd'),
113   ('a', 'b', 'e'),
114   ('a', 'c', 'd'),
115   ('a', 'c', 'e'), ...
116
117 # 重複を許す組み合わせ
118 list(itertools.combinations_with_replacement(A, 3))
119 [('a', 'a', 'a'),
120   ('a', 'a', 'b'),
121   ('a', 'a', 'c'),
122   ('a', 'b', 'b'),
123   ('a', 'b', 'c'), ...
124
125
126 ##### 編集距離 #####
127 S1 = 'yafo'
128 S2 = 'yahoo'
129 def levenshtein(s1,s2):
130     n, m = len(s1), len(s2)
131     dp = [[0]*(m+1) for _ in range(n + 1)]
132
133     for i in range(n+1):dp[i][0] = i
134     for j in range(m+1):dp[0][j] = j
135
136     for i in range(1, n + 1):
137         for j in range(1, m + 1):
138             if s1[i-1] == s2[j-1]:cost = 0
139             else:cost = 1
140             dp[i][j] = min(dp[i - 1][j] + 1,          # insertion
141                           dp[i][j - 1] + 1,          # deletion
142                           dp[i - 1][j - 1] + cost)    # replacement
143     return dp[n][m]
144 print(levenshtein(S1,S2))
145

```