How to Generate All Combinations

> Generating all combinations is a interesting problem, I will discuss three algorithms to solve that, the first two are form TAOCP, the third is from https://graphics.stanford.edu/~seander/bithacks.html.

Problem:

Generate all the combinations (represented by bit string) of $\binom{n}{t}$ in lexicographic order.

We use c to represent the positions of 1, such as :

```
000111 # c[3]c[2]c[1]=210
001011 # c[3]c[2]c[1]=310
001101 # c[3]c[2]c[1]=320
```

Naive Nested method:

The idea of the first method is using nested loops, which is easy to understand, but it only works if t is very small.

```
for c[3] = 2, 3,..., n-1
  for c[2] = 1, 2,..., c[3]-1
    for c[1]= 0 , 1,...,c[2]-1
      visit (c[3],c[2],c[1])
```

Lexicographic combinations:

The idea of the second algorithm is keeping moving the 1 bit to generate, but it can work when t is large.

```
#Initialize
c[t+1]=n # it shows the length og the bit string
c[t+2]=0
for 1<=j<=t
  c[j]=j-1

While (j<=t)
  visit (c[t],...,c[1])
  j=1
  while c[j]+1 == c[j+1]
    c[j]=j-1 # if the first bit change, it should recover in next enumeration
    j=j+1
  c[j]++

```

Bit Hack:

The core idea is move the first 1 appears after the rightmost (not last) 0 to the left and move all the 1 appear after that 1 to the rightmost !

```
def next_perm(v): #generate next permutations
```

```
    t = (v | ( v - 1)) + 1
    w = t | ((((t & -t) / (v & -v)) >> 1) - 1)
    return w
```

v | (v - 1) let the rightmost bit become 1, so if we add 1, the target 1 will move to the left, and set 0 for all the 1 on it's right, so we have the left part of the bit string.

x & -x isolate the rightmost 1 bit, because we want to know the remaining 1's position on the right part, so we calculate the last 1's position of the left part and the last 1's position of the right part, the form here is xxx10|11xxx, so there are position is xxxxx|xxx100 - 1 = xxxxx|xxx11. The division here is to let the left part 1 move to the right-end and caculate the number, right shift is to minus the "0" bit.

Add them up, so we get the next permutation.