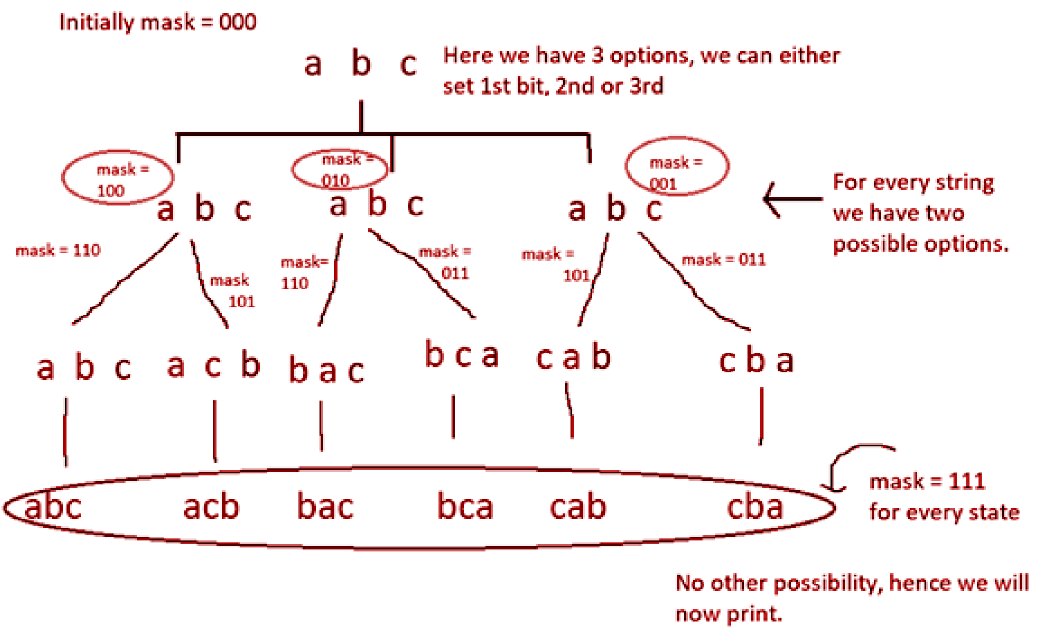
Permutations Generation

We saw how to generate the subsets of a set using bitmasking. Now we will see how to generate all possible permutations using bitmasks.

Problem Statement: Given a string we want to generate all the permutations of it.
Example: String : abc
Permutations:
abc
acb
bac
bca
cab
cba

Solution : Here, we will create a mask for every possible state and once every character is visited we'll print our answer. Length of the mask will obviously be equal to the length of the string.
At every iteration we'll be checking which all bits are not set and then try different permutations by setting different bits in every iteration.
If a particular bit is already set then we need not do anything, but if it isn't then we've to try all possible combinations. Different states of the mask will decide what the next combination is going to be.


Initially mask = 000
a b c
Here we have 3 options, we can either set 1st bit, 2nd or 3rd
mask = 100
mask = 010
mask = 001
a b c
a b c
a b c
For every string we have two possible options.
mask = 110
mask 101
mask= 110
mask = 011
mask = 101
mask = 011
a b c
a c b
b a c
b c a
c a b
c b a
abc
acb
bac
bca
cab
cba
mask = 111 for every state
No other possibility, hence we will now print.

In the above figure, the path to the permutation acb looks as follows:

- In the first step character, a is selected which makes the first bit of our mask set.

- At the second step, we choose c to be the next character, hence the 3rd position, i.e the original position of c is set. Now are string reads ac and our mask becomes 101 (5).
- Finally, we have only one unset position in the mask, hence we append character b to our string and finally print the permutation since our mask now becomes 111 = ((2^3) - 1).

Let us now look at the code for this problem.

```
function generatePermutations(S, N, mask, perm)
        /*
                We are passing the original string, its length,
                mask of visited characters and current permutation
        */

        //   Base Case of all characters included

        if mask == 2^N - 1
                print perm
                return

        for i = 0 to N - 1

                //   The ith character is already not included
                if (mask>>i & 1) == 0
                        perm = perm + S[i]
                        //   mark the current position and generate next permutations
                        generatePermutations(S, N, mask|1<<i, perm)

                        //   remove the last character
                        perm.pop_back()

        return
```