Last update: June 8, 2022 Translated From: e-maxx.ru

# Generating all $K$-combinations

In this article we will discuss the problem of generating all $K$-combinations. Given the natural numbers $N$ and $K$, and considering a set of numbers from $1$ to $N$. The task is to derive all **subsets of size** $K$.

## Generate next lexicographical $K$-combination

First we will generate them in lexicographical order. The algorithm for this is simple. The first combination will be $1, 2, \ldots, K$. Now let's see how to find the combination that immediately follows this, lexicographically. To do so, we consider our current combination, and find the rightmost element that has not yet reached its highest possible value. Once finding this element, we increment it by $1$, and assign the lowest valid value to all subsequent elements.

```
bool next_combination(vector<int>& a, int n) {
    int k = (int)a.size();
    for (int i = k - 1; i >= 0; i--) {
        if (a[i] < n - k + i + 1) {
            a[i]++;
            for (int j = i + 1; j < k; j++)
                a[j] = a[j - 1] + 1;
            return true;
        }
    }
    return false;
}
```

## Generate all $K$-combinations such that adjacent combinations differ by one element

This time we want to generate all $K$-combinations in such an order, that adjacent combinations differ exactly by one element.

This can be solved using the Gray Code: If we assign a bitmask to each subset, then by generating and iterating over these bitmasks with Gray codes, we can obtain our answer.

The task of generating $K$-combinations can also be solved using Gray Codes in a different way: Generate Gray Codes for the numbers from $0$ to $2^N - 1$ and leave only those codes containing $K$ $1$s. The surprising fact is that in the resulting sequence of $K$ set bits, any two

neighboring masks (including the first and last mask - neighboring in a cyclic sense) - will differ exactly by two bits, which is our objective (remove a number, add a number).

Let us prove this:

For the proof, we recall the fact that the sequence $G(N)$ (representing the $N^{\text{th}}$ Gray Code) can be obtained as follows:

$$G(N) = 0G(N-1) \cup 1G(N-1)^{\mathrm{R}}$$

That is, consider the Gray Code sequence for $N-1$, and prefix $0$ before every term. And consider the reversed Gray Code sequence for $N-1$ and prefix a $1$ before every mask, and concatenate these two sequences.

Now we may produce our proof.

First, we prove that the first and last masks differ exactly in two bits. To do this, it is sufficient to note that the first mask of the sequence $G(N)$, will be of the form $N-K$ $0$s, followed by $K$ $1$s. As the first bit is set as $0$, after which $(N-K-1)$ $0$s follow, after which $K$ set bits follow and the last mask will be of the form $1$, then $(N-K)$ $0$s, then $K-1$ $1$s. Applying the principle of mathematical induction, and using the formula for $G(N)$, concludes the proof.

Now our task is to show that any two adjacent codes also differ exactly in two bits, we can do this by considering our recursive equation for the generation of Gray Codes. Let us assume the content of the two halves formed by $G(N-1)$ is true. Now we need to prove that the new consecutive pair formed at the junction (by the concatenation of these two halves) is also valid, i.e. they differ by exactly two bits.

This can be done, as we know the last mask of the first half and the first mask of the second half. The last mask of the first half would be $1$, then $(N-K-1)$ $0$s, then $K-1$ $1$s. And the first mask of the second half would be $0$, then $(N-K-2)$ $0$s would follow, and then $K$ $1$s. Thus, comparing the two masks, we find exactly two bits that differ.

The following is a naive implementation working by generating all $2^n$ possible subsets, and finding subsets of size $K$.

```
int gray_code (int n) {
    return n ^ (n >> 1);
}

int count_bits (int n) {
    int res = 0;
    for (; n; n >>= 1)
        res += n & 1;
    return res;
}

void all_combinations (int n, int k) {
```

```
    for (int i = 0; i < (1 << n); i++) {
        int cur = gray_code (i);
        if (count_bits(cur) == k) {
            for (int j = 0; j < n; j++) {
                if (cur & (1 << j))
                    cout << j + 1;
            }
            cout << "\n";
        }
    }
}
```

It's worth mentioning that a more efficient implementation exists that only resorts to building valid combinations and thus works in $O\left(N \cdot \binom{N}{K}\right)$ however it is recursive in nature and for smaller values of $N$ it probably has a larger constant than the previous solution.

The implementation is derived from the formula:

$$G(N, K) = 0G(N-1, K) \cup 1G(N-1, K-1)^{\mathrm{R}}$$

This formula is obtained by modifying the general equation to determine the Gray code, and works by selecting the subsequence from appropriate elements.

Its implementation is as follows:

```
vector<int> ans;

void gen(int n, int k, int idx, bool rev) {
    if (k > n || k < 0)
        return;

    if (!n) {
        for (int i = 0; i < idx; ++i) {
            if (ans[i])
                cout << i + 1;
        }
        cout << "\n";
        return;
    }

    ans[idx] = rev;
    gen(n - 1, k - rev, idx + 1, false);
    ans[idx] = !rev;
    gen(n - 1, k - !rev, idx + 1, true);
}

void all_combinations(int n, int k) {
    ans.resize(n);
    gen(n, k, 0, false);
}
```

Contributors:

RameshAditya (91.49%) jakobkogler (6.38%) adamant-pwn (2.13%)