1415. The k-th Lexicographical String of All Happy Strings of Length n

A **happy string** is a string that:

- consists only of letters of the set `['a', 'b', 'c']`.
- `s[i] != s[i + 1]` for all values of `i` from `1` to `s.length - 1` (string is 1-indexed).

For example, strings **"abc", "ac", "b"** and **"abcbabcbcb"** are all happy strings and strings **"aa", "baa"** and **"ababbc"** are not happy strings.

Given two integers `n` and `k`, consider a list of all happy strings of length `n` sorted in lexicographical order.

Return *the kth string* of this list or return an **empty string** if there are less than `k` happy strings of length `n`.

Example 1:

```
Input: n = 1, k = 3
Output: "c"
Explanation: The list ["a", "b", "c"] contains all happy strings of length 1. The
third string is "c".
```

Example 2:

```
Input: n = 1, k = 4
Output: ""
Explanation: There are only 3 happy strings of length 1.
```

Example 3:

```
Input: n = 3, k = 9
Output: "cab"
Explanation: There are 12 different happy string of length 3 ["aba", "abc", "aca",
"acb", "bab", "bac", "bca", "bcb", "cab", "cac", "cba", "cbc"]. You will find the
9th string = "cab"
```

Constraints:

- `1 <= n <= 10`
- `1 <= k <= 100`

1415. The k-th Lexicographical String of All Happy Strings of Length n

Overview

We are given a positive integer n, which represents the length of the string, and an integer k. Our task is to find the $k-th$ happy string of length n when all *happy* strings are listed in lexicographical order. Let’s break this down:

- **Happy Strings**: A string is called happy if it consists only of the characters `'a'`, `'b'`, and `'c'`, and no two consecutive characters are the same. For example, `"abc"` and `"aba"` are happy strings, but `"aa"` and `"ad"` are not.
- **Lexicographical Order**: This is the order in which words appear in a dictionary. When comparing two strings, we look at the first different character. The one with the smaller character (closer to `'a'` in the alphabet) comes first. For example, `"abc"` comes before `"acb"` because `'b'` comes before `'c'`.

Note: If there are fewer than k such strings, we return an empty string.

1415. The k-th Lexicographical String of All Happy Strings of Length n

```
/*
    Recursive+backtracking: Generate all happy strings
    Time complexity:  O(n.2^n)
    Space complexity:  O(n)
*/
class Solution {
    public:
        std::string getHappyString(int n, int k) {
            int cnt=0; // index in sorted order list of happy strings
            std::string cur; // Generated happy string
            std::string ans; // Final answer

            // Function to generate happy strings in lexicographically order
            auto solve=[&](auto& self)->void{
                // If we generate a happy string of size n
                if(cur.size()==n){
                    cnt++; // Increment the counter (its index in sorted order list of happy strings)
                    if(cnt==k) ans=cur; // If we read the k-th happy string, we get our answer
                    return;
                }

                // For each happy string, we use the letters 'a','b' and 'c'
                for(char c='a';c<='c';++c){
                    // If the generated string is not happy(cur[i-1]==c), go next letter
                    if(!cur.empty() && cur.back()==c) continue;

                    // Otherwise
                    cur.push_back(c); // Push the current letter to the current happy string
                    self(self); // Continue generating
                    if(!ans.empty()) return; // If we get our result, stop the process
                    cur.pop_back(); // Backtrack by removing the last character
                }
            };

            solve(solve);

            return ans;

        }
};
```

1415. The k-th Lexicographical String of All Happy Strings of Length n

```
/*
  Recursive+backtracking: Generate all happy strings of the group where the k-th
  happy string belongs
  Time complexity:  O(2^n)
  Space complexity:  O(n)
*/
class Solution {
  public:
    std::string getHappyString(int n, int k) {
      // Count total happy string in each group
      int group_count=1<<(n-1);

      // Count total number of happy strings
      int happy_count=3*group_count;

      // If k exceeds the total number of happy strings
      if(k>happy_count) return "";

      // Determine which group belongs the k-th happy string ('a','b' or 'c')
      int group=ceil(k*1.0/group_count*1.0);

      // Update k of the k-th happy string in the its group
      k=k-(group-1)*group_count;

      int cnt=0; // index in sorted order list of happy strings

      std::string cur; // Generated happy string
      cur.push_back((group-1)+'a'); // We know, which group belong the k-th happy string

      std::string ans; // Final answer
```

```
            // Function to generate happy strings in lexicographically order
            auto solve=[&](auto& self)->void{
                // If we generate a happy string of size n
                if(cur.size()==n){
                    cnt++; // Increment the counter (its index in sorted order list of happy strings)
                    if(cnt==k) ans=cur; // If we read the k-th happy string, we get our answer
                    return;
                }

                // For each happy string, we use the letters 'a','b' and 'c'
                for(char c='a';c<='c';++c){
                    // If the generated string is not happy(cur[i-1]==c), go next letter
                    if(cur.back()==c) continue;

                    // Otherwise
                    cur.push_back(c); // Push the current letter to the current happy string
                    self(self); // Continue generating
                    if(!ans.empty()) return; // If we get our result, stop the process
                    cur.pop_back(); // Backtrack by removing the last character
                }
            };

            solve(solve);

            return ans;

        }
};
```

1415. The k-th Lexicographical String of All Happy Strings of Length n

```
/*
   Math: Select and shrink the group of the k-th happy string
   Time complexity: O(n)
   Space complexity: O(1)
*/
class Solution {
   public:
      std::string getHappyString(int n, int k) {
         // Count total happy string in each group
         int group_count=1<<(n-1);

         // Count total number of happy strings
         int happy_count=3*group_count;

         // If k exceeds the total number of happy strings
         if(k>happy_count) return "";

         // Determine which group belongs the k-th happy string ('a','b' or 'c')
         int group=ceil(k*1.0/group_count*1.0);

         // Update k of the k-th happy string in the its group
         k=k-(group-1)*group_count;

         std::string ans; // Generated happy string
         ans.push_back((group-1)+'a'); // We know, which group belong the k-th happy string

         // Map each group 'a','b' and 'c' with its two options
         std::vector<std::vector<char>> mapping={{'b','c'},{'a','c'},{'a','b'}};

         // Build the remaining answer letters, by going down in the tree and
         // selecting the correspondent group of the k-th happy string
         for(int i=1;i<n;++i){
            group_count/=2;
            group=ceil(k*1.0/group_count*1.0);
            k=k-(group-1)*group_count;
            ans+=mapping[ans.back()-'a'][group-1];
         }

         return ans;

      }
};
```