

41. 缺失的第一个正数

41. 缺失的第一个正数

难度 **困难**  463  收藏  分享  切换为英文  关注  反馈

给你一个未排序的整数数组，请你找出其中没有出现的最小的正整数。

示例 1:

输入: [1,2,0]

输出: 3

示例 2:

输入: [3,4,-1,1]

输出: 2

示例 3:

输入: [7,8,9,11,12]

输出: 1

提示:

你的算法的时间复杂度应为 $O(n)$ ，并且只能使用常数级别的额外空间。

```
class Solution{
public:
    int firstMissingPositive(vector<int>& nums)
    {
        int n = nums.size();

        for(int i = 0; i < n; ++ i)
            while(nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] != nums[i])
                swap(nums[i], nums[nums[i] - 1]);

        for(int i = 0; i < n; ++ i)
            if(nums[i] != i + 1)
                return i + 1;

        return n + 1;
    }
};
```

42. 接雨水

42. 接雨水

难度 困难

👍 1171

📖 收藏

🗨 分享

🌐 切换为英文

🔔 关注

📄 反馈

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。



上面是由数组 `[0,1,0,2,1,0,1,3,2,1,2,1]` 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。感谢 Marcos 贡献此图。

示例:

输入: `[0,1,0,2,1,0,1,3,2,1,2,1]`

输出: 6

```
class Solution {
public:
    int trap(vector<int>& height) {
        int res = 0;
        stack<int> stk;

        for(int i = 0; i < height.size(); i++)
        {
            int last = 0;
            while(stk.size() && height[stk.top()] <= height[i])
            {
                int t = stk.top();
                stk.pop();
                res += (i - t - 1) * (height[t] - last);
                last = height[t];
            }

            if(stk.size()) res += (i - stk.top() - 1) * (height[i] - last);
            stk.push(i);
        }
        return res;
    }
};
```

43. 字符串相乘

43. 字符串相乘

难度 中等 310 收藏 分享 切换为英文 关注 反馈

给定两个以字符串形式表示的非负整数 `num1` 和 `num2`，返回 `num1` 和 `num2` 的乘积，它们的乘积也表示为字符串形式。

示例 1:

输入: `num1 = "2"`, `num2 = "3"`
输出: `"6"`

示例 2:

输入: `num1 = "123"`, `num2 = "456"`
输出: `"56088"`

说明:

1. `num1` 和 `num2` 的长度小于110。
2. `num1` 和 `num2` 只包含数字 0-9。
3. `num1` 和 `num2` 均不以零开头，除非是数字 0 本身。
4. 不能使用任何标准库的大数类型（比如 `BigInteger`）或直接将输入转换为整数来处理。

```
class Solution {
public:
    string multiply(string num1, string num2) {
        int n = num1.length(), m = num2.length();
        vector<int> a(n), b(m), c(n + m);
        for (int i = 0; i < n; i++)
            a[n - i - 1] = num1[i] - '0';
        for (int i = 0; i < m; i++)
            b[m - i - 1] = num2[i] - '0';

        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++) {
                c[i + j] += a[i] * b[j];
                c[i + j + 1] += c[i + j] / 10;
                c[i + j] %= 10;
            }

        int l = n + m;
        while (l > 1 && c[l - 1] == 0) l--;
        string ans = "";
        for (int i = l - 1; i >= 0; i--)
            ans += c[i] + '0';
        return ans;
    }
};
```

44. 通配符匹配

44. 通配符匹配

难度 困难 314 收藏 分享 切换为英文 关注 反馈

给定一个字符串 (s) 和一个字符模式 (p)，实现一个支持 '?' 和 '*' 的通配符匹配。

- '?' 可以匹配任何单个字符。
- '*' 可以匹配任意字符串（包括空字符串）。

两个字符串完全匹配才算匹配成功。

说明:

- s 可能为空，且只包含从 $a-z$ 的小写字母。
- p 可能为空，且只包含从 $a-z$ 的小写字母，以及字符 ? 和 *。

示例 1:

输入：
 $s = "aa"$
 $p = "a"$
输出: false
解释: "a" 无法匹配 "aa" 整个字符串。

示例 2:

输入：
 $s = "aa"$
 $p = "**"$
输出: true
解释: '*' 可以匹配任意字符串。

示例 3:

输入：
 $s = "cb"$
 $p = "?a"$
输出: false
解释: '?' 可以匹配 'c'，但第二个 'a' 无法匹配 'b'。

```
class Solution {
public:
    bool isMatch(string s, string p) {
        int n = s.length(), m = p.length();

        vector<vector<bool>> f(n + 1, vector(m + 1, false));

        f[0][0] = true;

        for (int i = 0; i <= n; i++)
            for (int j = 1; j <= m; j++) {
                char y = p[j - 1];
                if (i > 0) {
                    char x = s[i - 1];
                    if (x == y || y == '?')
                        f[i][j] = f[i][j] | f[i - 1][j - 1];
                }
            }
    }
};
```


```

        }
        if (y == '*') {
            f[i][j] = f[i][j] | f[i][j - 1];
            if (i > 0)
                f[i][j] = f[i][j] | f[i - 1][j];
        }
    }
    return f[n][m];
}
};

```

45. 跳跃游戏 II

45. 跳跃游戏 II

难度 **困难**  417  收藏  分享  切换为英文  关注  反馈

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

你的目标是使用最少的跳跃次数到达数组的最后一个位置。

示例:

输入: [2,3,1,1,4]

输出: 2

解释: 跳到最后一个位置的最小跳跃数是 2。

从下标为 0 跳到下标为 1 的位置，跳 1 步，然后跳 3 步到达数组的最后一个位置。

说明:

假设你总是可以到达数组的最后一个位置。

```

class Solution {
public:
    int jump(vector<int>& nums) {
        int n = nums.size();
        vector<int> f(n);
        f[0] = 0;
        int last = 0;
        for (int i = 1; i < n; i++) { // 依次求f[i]的值。
            while (i > last + nums[last]) // 根据i来更新last。
                last++;
            f[i] = f[last] + 1; // 根据f[last]更新f[i]。
        }
        return f[n - 1];
    }
};

```

46. 全排列

46. 全排列

难度 中等

👍 602

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个没有重复数字的序列，返回其所有可能的全排列。

示例:

```
输入: [1,2,3]
输出:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

```
class Solution {
public:
    vector<vector<int>> ans;
    vector<bool> st;
    vector<int> path;

    vector<vector<int>> permute(vector<int>& nums) {
        for (int i = 0; i < nums.size(); i++) st.push_back(false);
        dfs(nums, 0);
        return ans;
    }

    void dfs(vector<int> &nums, int u)
    {
        if (u == nums.size())
        {
            ans.push_back(path);
            return ;
        }

        for (int i = 0; i < nums.size(); i++)
            if (!st[i])
            {
                st[i] = true;
                path.push_back(nums[i]);
                dfs(nums, u + 1);
                st[i] = false;
                path.pop_back();
            }
    }
};
```

47. 全排列 II

47. 全排列 II

难度 中等

👍 261

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

🗉 反馈

给定一个可包含重复数字的序列，返回所有不重复的全排列。

示例:

```
输入: [1,1,2]
输出:
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

```
class Solution {
public:
    vector<bool> st;
    vector<int> path;
    vector<vector<int>> ans;

    vector<vector<int>> permuteUnique(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        st = vector<bool>(nums.size(), false);
        path = vector<int>(nums.size());
        dfs(nums, 0, 0);
        return ans;
    }

    void dfs(vector<int>& nums, int u, int start)
    {
        if (u == nums.size())
        {
            ans.push_back(path);
            return;
        }

        for (int i = start; i < nums.size(); i++)
            if (!st[i])
            {
                st[i] = true;
                path[u] = nums[i];
                if (u + 1 < nums.size() && nums[u + 1] != nums[u])
                    dfs(nums, u + 1, 0);
                else
                    dfs(nums, u + 1, i + 1);
                st[i] = false;
            }
    }
};
```

48. 旋转图像

48. 旋转图像

难度 中等

👍 414

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

🗉 反馈

给定一个 $n \times n$ 的二维矩阵表示一个图像。

将图像顺时针旋转 90 度。

说明：

你必须在原地旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要使用另一个矩阵来旋转图像。

示例 1:

```
给定 matrix =  
[  
  [1,2,3],  
  [4,5,6],  
  [7,8,9]  
],
```

原地旋转输入矩阵，使其变为：

```
[  
  [7,4,1],  
  [8,5,2],  
  [9,6,3]  
]
```

```
class Solution {  
public:  
    void rotate(vector<vector<int>>& matrix) {  
        int n = matrix.size();  
        for(int i = 0; i < n; i++)  
            for(int j = i + 1; j < n; j++)  
                swap(matrix[i][j], matrix[j][i]);  
  
        for(int i = 0; i < n; i++)  
            for(int j = 0, k = n - 1; j < k; j++, k--)  
                swap(matrix[i][j], matrix[i][k]);  
    }  
};
```

49. 字母异位词分组

49. 字母异位词分组

难度 中等

👍 319

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个字符串数组，将字母异位词组合在一起。字母异位词指字母相同，但排列不同的字符串。

示例:

```
输入: ["eat", "tea", "tan", "ate", "nat", "bat"],
输出:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

说明:

- 所有输入均为小写字母。
- 不考虑答案输出的顺序。

通过次数 44,410 | 提交次数 105,501

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string,vector<string>> hash;
        for(auto &str : strs)
        {
            string key = str;
            sort(key.begin(),key.end());
            hash[key].push_back(str);
        }

        vector<vector<string>> res;
        for(auto item : hash) res.push_back(item.second);

        return res;
    }
};
```

50. Pow(x, n)

50. Pow(x, n)

难度 中等

👍 299

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

实现 $\text{pow}(x, n)$ ，即计算 x 的 n 次幂函数。

示例 1:

输入: 2.00000, 10
输出: 1024.00000

示例 2:

输入: 2.10000, 3
输出: 9.26100

示例 3:

输入: 2.00000, -2
输出: 0.25000
解释: $2^{-2} = 1/2^2 = 1/4 = 0.25$

说明:

- $-100.0 < x < 100.0$
- n 是 32 位有符号整数，其数值范围是 $[-2^{31}, 2^{31} - 1]$ 。

```
class Solution {
public:
    #define LL long long
    double myPow(double x, int n) {
        double ans = 1, p = x;
        LL t = abs((LL)(n)); // 注意 x 为 INT_MIN 时, abs 会爆掉 int。
        for (; t >= 1; t >>= 1) { // 将 n 进行二进制拆分。
            if (t & 1)
                ans = ans * p; // p 是提前计算好的批量连乘。
            p = p * p; // 每次更新 p, 自身平方。
        }
        return n > 0 ? ans : 1 / ans;
    }
};
```

51. N皇后

51. N皇后

难度 困难

👍 372

❤ 收藏

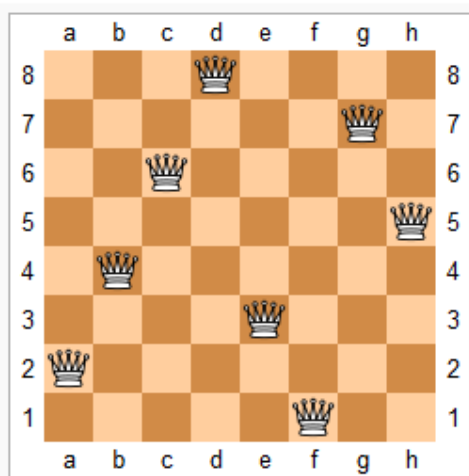
🔗 分享

🌐 切换为英文

👤 关注

📝 反馈

n 皇后问题研究的是如何将 n 个皇后放置在 $n \times n$ 的棋盘上，并且使皇后彼此之间不能相互攻击。



One solution to the eight queens puzzle

上图为 8 皇后问题的一种解法。

给定一个整数 n ，返回所有不同的 n 皇后问题的解决方案。

每一种解法包含一个明确的 n 皇后问题的棋子放置方案，该方案中 'Q' 和 '.' 分别代表了皇后和空位。

示例:

```
输入: 4
输出: [
  [".Q..", // 解法 1
   "...Q",
   "Q...",
   "..Q."],

  [ "..Q.", // 解法 2
    "Q...",
    "...Q",
    "..Q." ]]
```

```
class Solution {
public:
    vector<vector<string>> ans;
    vector<string> path;
    vector<bool> row, col, diag, anti_diag;

    vector<vector<string>> solveNQueens(int n) {
        row = col = vector<bool>(n, false);
        diag = anti_diag = vector<bool>(2 * n, false);
        path = vector<string>(n, string(n, '.'));
        dfs(0, 0, 0, n);
        return ans;
    }

    void dfs(int x, int y, int s, int n)
    {
```

```

    if (y == n) x ++ , y = 0;
    if (x == n)
    {
        if (s == n) ans.push_back(path);
        return ;
    }

    dfs(x, y + 1, s, n);
    if (!row[x] && !col[y] && !diag[x + y]
        && !anti_diag[n - 1 - x + y])
    {
        row[x] = col[y] = diag[x + y] = anti_diag[n - 1 - x + y] = true;
        path[x][y] = 'Q';
        dfs(x, y + 1, s + 1, n);
        path[x][y] = '.';
        row[x] = col[y] = diag[x + y] = anti_diag[n - 1 - x + y] = false;
    }
}
};

```

52. N皇后 II

52. N皇后 II

难度 困难

👍 108

📖 收藏

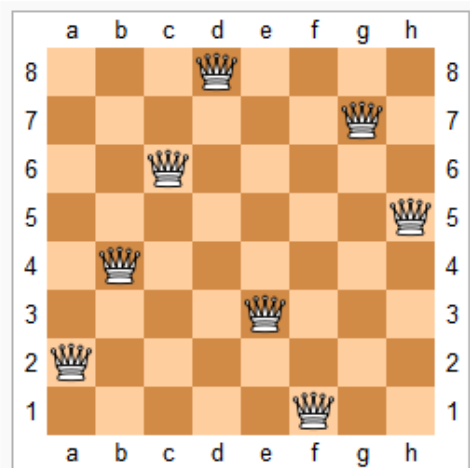
🔗 分享

🌐 切换为英文

👤 关注

📝 反馈

n 皇后问题研究的是如何将 n 个皇后放置在 $n \times n$ 的棋盘上，并且使皇后彼此之间不能相互攻击。



One solution to the eight queens puzzle

上图是 8 皇后问题的一种解法。

给定一个整数 n ，返回 n 皇后不同的解决方案的数量。

示例:

输入: 4

输出: 2

解释: 4 皇后问题存在如下两个不同的解法。

```
[
  [".Q..", // 解法 1
   "...Q",
   "Q...",
   "..Q."],

  [ "..Q.", // 解法 2
   "Q...",
   "...Q",
   ".Q.."]
]
```

1

```
class Solution {
public:

    int ans = 0, n;
    vector<bool> col, d, ud;
    int totalNQueens(int _n) {
        n = _n;
        col = vector<bool>(n);
        d = ud = vector<bool>(n * 2);
        dfs(0);

        return ans;
    }

    void dfs(int u)
    {
```

```

        if(u == n )
        {
            ans ++;
            return;
        }

        for(int i = 0;i < n;i ++)
            if(!col[i] && !d[u + i] && !ud[u - i + n])
            {
                col[i] = d[u + i] = ud[u - i + n] = true;
                dfs(u + 1);
                col[i] = d[u + i] = ud[u - i + n] = false;
            }
    }
};

```

53. 最大子序和

53. 最大子序和

难度 **简单** 1812 收藏 分享 切换为英文 关注 反馈

给定一个整数数组 `nums`，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

示例:

输入: `[-2,1,-3,4,-1,2,1,-5,4]`,
 输出: `6`
 解释: 连续子数组 `[4,-1,2,1]` 的和最大，为 `6`。

进阶:

如果你已经实现复杂度为 $O(n)$ 的解法，尝试使用更为精妙的分治法求解。

```

class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int res = INT_MIN, last = 0;
        for(int i = 0; i < nums.size(); i++)
        {
            int now = max(last, 0) + nums[i];
            res = max(res, now);
            last = now;
        }

        return res;
    }
};

```

54. 螺旋矩阵

54. 螺旋矩阵

难度 中等

👍 340

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个包含 $m \times n$ 个元素的矩阵 (m 行, n 列), 请按照顺时针螺旋顺序, 返回矩阵中的所有元素。

示例 1:

```
输入:
[
  [ 1, 2, 3 ],
  [ 4, 5, 6 ],
  [ 7, 8, 9 ]
]
输出: [1,2,3,6,9,8,7,4,5]
```





示例 2:

```
输入:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
输出: [1,2,3,4,8,12,11,10,9,5,6,7]
```

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> res;
        int dx[] = {0,-1,0,1},dy[] = {1,0,-1,0};
        int x = 0,y = 0,d = 0;
        int n = matrix.size(),m = matrix[0].size();
        bool st[n][m] = {false};
        for(int i = 1;i <= n * m;i ++){
            int nx = x + dx[d],ny = y + dy[d];
            if(nx < 0 || nx >= n || ny < 0 || ny >= m || st[nx][ny]){
                d = (d + 1) % 4;
                nx = x + dx[d];
                ny = y + dy[d];
            }
            res.push_back(matrix[x][y]);
            st[x][y] = true;
            x = nx;
            y = ny;
        }
        return res;
    }
};
```

55. 跳跃游戏

55. 跳跃游戏

难度 中等  545  收藏  分享  切换为英文  关注  反馈

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

示例 1:

输入: [2,3,1,1,4]

输出: true

解释: 我们可以先跳 1 步，从位置 0 到达 位置 1，然后再从位置 1 跳 3 步到达最后一个位置。

示例 2:

输入: [3,2,1,0,4]

输出: false

解释: 无论如何，你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0，所以你永远不可能到达最后一个位置。

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int n = nums.size();
        int last = 0;
        for (int i = 1; i < n; i++) {
            while (last < i && i > last + nums[last])
                last++;
            if (last == i)
                return false;
        }
        return true;
    }
};
```

56. 合并区间

56. 合并区间

难度 中等

👍 349

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💡 反馈

给出一个区间的集合，请合并所有重叠的区间。

示例 1:

输入: `[[1,3],[2,6],[8,10],[15,18]]`

输出: `[[1,6],[8,10],[15,18]]`

解释: 区间 `[1,3]` 和 `[2,6]` 重叠，将它们合并为 `[1,6]`。

示例 2:

输入: `[[1,4],[4,5]]`

输出: `[[1,5]]`

解释: 区间 `[1,4]` 和 `[4,5]` 可被视为重叠区间。

```
class Solution {
public:
    static bool cmp(const Interval& x, const Interval& y) {
        if (x.start != y.start)
            return x.start < y.start;
        return x.end < y.end;
    }
    vector<Interval> merge(vector<Interval>& intervals) {
        int n = intervals.size();
        sort(intervals.begin(), intervals.end(), cmp);
        vector<Interval> ans;
        if (n == 0)
            return ans;
        Interval cur = intervals[0];
        for (int i = 1; i < n; i++) {
            if (intervals[i].start > cur.end) {
                ans.push_back(cur);
                cur = intervals[i];
            }
            else if (intervals[i].end > cur.end)
                cur.end = intervals[i].end;
        }
        ans.push_back(cur);
        return ans;
    }
};
```

57. 插入区间

57. 插入区间

难度 **困难** 127 收藏 分享 切换为英文 关注 反馈

给出一个无重叠的，按照区间起始端点排序的区间列表。

在列表中插入一个新的区间，你需要确保列表中的区间仍然有序且不重叠（如果有必要的话，可以合并区间）。

示例 1:

```
输入: intervals = [[1,3],[6,9]], newInterval = [2,5]
输出: [[1,5],[6,9]]
```

示例 2:

```
输入: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
输出: [[1,2],[3,10],[12,16]]
解释: 这是因为新的区间 [4,8] 与 [3,5],[6,7],[8,10] 重叠。
```

58. 最后一个单词的长度

58. 最后一个单词的长度

难度 **简单** 188 收藏 分享 切换为英文 关注 反馈

给定一个仅包含大小写字母和空格 ' ' 的字符串 `s`，返回其最后一个单词的长度。如果字符串从左向右滚动显示，那么最后一个单词就是最后出现的单词。

如果不存在最后一个单词，请返回 0。

说明：一个单词是指仅由字母组成、不包含任何空格字符的 **最大子字符串**。

示例:

```
输入: "Hello World"
输出: 5
```

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int sum = 0, res = 0;
        for (int i = 0; i < s.size(); i++)
            if (s[i] == ' ')
            {
                if (sum) res = sum;
                sum = 0;
            }
            else
                sum++;
        if (sum) res = sum;
        return res;
    }
};
```

59. 螺旋矩阵 II

59. 螺旋矩阵 II

难度 中等  173     

给定一个正整数 n ，生成一个包含 1 到 n^2 所有元素，且元素按顺时针顺序螺旋排列的正方形矩阵。

示例:

```
输入: 3
输出:
[
  [ 1, 2, 3 ],
  [ 8, 9, 4 ],
  [ 7, 6, 5 ]
]
```

```
class Solution {
public:
    vector<vector<int>> generateMatrix(int n) {
        vector<vector<int>> res(n, vector<int>(n, 0));

        int dx[4] = {-1, 0, 1, 0};
        int dy[4] = {0, 1, 0, -1};

        for (int x = 0, y = 0, i = 0, d = 1; i < n * n; i++) {
            res[x][y] = i + 1;
            int a = x + dx[d], b = y + dy[d];
            if (a < 0 || a == n || b < 0 || b == n || res[a][b]) {
                d = (d + 1) % 4;
                a = x + dx[d], b = y + dy[d];
            }
            x = a, y = b;
        }

        return res;
    }
};
```

60. 第k个排列

60. 第k个排列

难度 中等  217     

给出集合 $[1, 2, 3, \dots, n]$ ，其所有元素共有 $n!$ 种排列。

按大小顺序列出所有排列情况，并一一标记，当 $n = 3$ 时，所有排列如下：

1. "123"
2. "132"
3. "213"
4. "231"
5. "312"
6. "321"

给定 n 和 k ，返回第 k 个排列。

说明：

- 给定 n 的范围是 $[1, 9]$ 。
- 给定 k 的范围是 $[1, n!]$ 。

示例 1:

输入： $n = 3, k = 3$
输出："213"

示例 2:

输入： $n = 4, k = 9$
输出："2314"

```
class Solution {
public:

    string getPermutation(int n, int k) {
        string res;
        vector<bool> st(n, false);
        for (int i = 0; i < n; i++) //从高位到低位依次枚举每一位
        {
            int f = 1;
            for (int j = 1; j <= n - i - 1; j++) f *= j; //计算 (n-i-1)!
            int next = 0;
            if (k > f) //确定当前位是第几个未使用过的数
            {
                int t = k / f;
                k %= f;
                if (k == 0) k = f, t--;
                while (t)
                {
                    if (!st[next]) t--;
                    next++;
                }
            }
            while (st[next]) next++;
            res += to_string(next + 1);
        }
    }
};
```

```
        st[next] = true;
    }

    return res;
}
};
```