# 题目

给你一个以字符串表示的非负整数 num 和一个整数 k ，移除这个数中的 k 位数字，使得剩下的数字最小。请你以字符串形式返回这个最小的数字。

示例 1 ：

输入：num = "1432219", k = 3

输出："1219"

解释：移除掉三个数字 4, 3, 和 2 形成一个新的最小的数字 1219 。

示例 2 ：

输入：num = "10200", k = 1

输出："200"

解释：移掉首位的 1 剩下的数字为 200. 注意输出不能有任何前导零。

示例 3 ：

输入：num = "10", k = 2

输出："0"

解释：从原数字移除所有的数字，剩余为空就是 0 。

提示：

1 <= k <= num.length <= 105

num 仅由若干位数字（0 - 9）组成

除了 0 本身之外，num 不含任何前导零

# 分析

## 方法一：贪心+单调栈

class Solution {

public:

string removeKdigits(string num, int k) {

string stack = ""; // 使用字符串来模拟栈

for (char digit : num) {

// 当栈不为空且 k 大于 0 且当前数字小于栈顶数字时，弹出栈顶数字

while (!stack.empty() && k > 0 && digit < stack.back()) {

stack.pop\_back();

k--;

}

// 如果当前数字不为零或者栈不为空（即不能删除前导零），则将当前数字入栈

if ((stack.empty() && digit != '0') || !stack.empty()) {

stack.push\_back(digit);

}

}

// 如果 k 大于 0，继续弹出栈顶元素

while (!stack.empty() && k > 0) {

stack.pop\_back();

k--;

}

// 如果结果字符串为空，返回 "0"；否则返回结果字符串

return stack.empty() ? "0" : stack;

}

};

执行错误（内存超限）：

class Solution {

public:

string removeKdigits(string num, int k) {

stack<char> st;

for (char digit : num) {

while (!st.empty() && k > 0 && digit < st.top()) {

st.pop();

k--;

}

st.push(digit);

}

// 如果还有剩余删除次数 k，继续弹出栈顶元素

while (!st.empty() && k > 0) {

st.pop();

k--;

}

// 构造结果字符串

string result = "";

while (!st.empty()) {

result = st.top() + result;

st.pop();

}

// 删除前导零

int start = 0;

while (start < result.length() && result[start] == '0') {

start++;

}

result = result.substr(start);

// 如果结果字符串为空，返回 "0"；否则返回结果字符串

return result.empty() ? "0" : result;

}

};

分析：

使用string不会出现内存超限，但是使用stack就会出现内存超限，问题分析。

在这种情况下，使用 `std::string` 和使用 `std::stack` 的内存使用差异可能是由于以下几个方面造成的：

1、内存分配方式不同：`std::string` 内部使用的是动态内存分配，会根据需要动态调整存储空间，而 `std::stack` 存储的是索引位置，不涉及动态内存分配。因此，使用 `std::stack` 可能会占用更多的内存空间。

2、字符串连接方式不同\*\*：在使用 `std::string` 的情况下，可以直接通过 `+` 运算符连接字符串，而在使用 `std::stack` 的情况下，需要先将字符入栈，最后再构造结果字符串。这种方式可能会导致额外的内存开销。

3、栈的实现方式\*\*：`std::stack` 的底层实现通常是基于其他数据结构（如 `std::deque` 或 `std::vector`），这些数据结构可能会引入额外的内存开销。

综上所述，使用 `std::string` 可能在内存使用上更加高效，而使用 `std::stack` 可能会引入额外的内存开销，导致内存超限。

做如下的优化：

class Solution {

public:

string removeKdigits(string num, int k) {

stack<char> s;

int n = num.size();

for (int i = 0; i < n; i++) {

// 如果当前栈非空，且还存在k，并且栈顶的元素大于当前数值，则出栈

while (!s.empty() && k > 0 && s.top() > num[i]) {

s.pop();

k--;

}

if (s.empty() && num[i] == '0')

continue;//跳过前置0

// 前面过滤后这里都入栈

s.push(num[i]);

}

string result;

while (!s.empty()) {

if (k > 0)//当还要再移除数字的时候：从此时单调递增栈的top部删去数字

k--;

else if (k == 0)//当不用再移除数字的时候：把字符串取出来到result

result += s.top();

s.pop();

}

reverse(result.begin(), result.end());//stl中的reverse函数

return result == "" ? "0" : result;

}

};