# 题目

有效 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导 0），整数之间用 '.' 分隔。

例如："0.1.2.201" 和 "192.168.1.1" 是 有效 IP 地址，但是 "0.011.255.245"、"192.168.1.312" 和 "192.168@1.1" 是 无效 IP 地址。

给定一个只包含数字的字符串 s ，用以表示一个 IP 地址，返回所有可能的有效 IP 地址，这些地址可以通过在 s 中插入 '.' 来形成。你 不能 重新排序或删除 s 中的任何数字。你可以按 任何 顺序返回答案。

示例 1：

输入：s = "25525511135"

输出：["255.255.11.135","255.255.111.35"]

示例 2：

输入：s = "0000"

输出：["0.0.0.0"]

示例 3：

输入：s = "101023"

输出：["1.0.10.23","1.0.102.3","10.1.0.23","10.10.2.3","101.0.2.3"]

提示：

1 <= s.length <= 20

s 仅由数字组成

# 分析

## 方法一：模拟

思路：

可以通过回溯法来解决。具体思路如下：

1、使用一个函数 isValidSegment 来判断一个字符串段是否是有效的 IP 地址的一部分。字符串段必须满足以下条件：

长度在 1 到 3 之间。

如果长度大于 1，则不能以 '0' 开头。

数值范围在 0 到 255 之间。

2、使用回溯法，从第一个字符开始，逐步尝试所有可能的分割方式，每次分割出一个合法的字符串段后，继续递归处理剩余部分。

3、当处理到第四段时，如果剩余部分可以分割为合法的字符串段，则将结果加入答案。

代码：

class Solution {

public:

vector<string> restoreIpAddresses(string s) {

vector<string> result;

vector<string> segments;

backtrack(result, s, 0, segments);

return result;

}

private:

void backtrack(vector<string>& result, const string& s, int start, vector<string>& segments) {

if (segments.size() == 4 && start == s.size()) {

result.push\_back(segments[0] + '.' + segments[1] + '.' + segments[2] + '.' + segments[3]);

return;

}

if (segments.size() == 4 || start == s.size()) {

return;

}

for (int i = 1; i <= 3 && start + i <= s.size(); ++i) {

string segment = s.substr(start, i);

if (isValidSegment(segment)) {

segments.push\_back(segment);

backtrack(result, s, start + i, segments);

segments.pop\_back();

}

}

}

bool isValidSegment(const string& segment) {

if (segment.size() > 1 && segment[0] == '0') {

return false;

}

int num = stoi(segment);

return num >= 0 && num <= 255;

}

};