1. Additive Number

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain **at least** three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

Given a string containing only digits '0'-'9', write a function to determine if it’s an additive number.

**Note:** Numbers in the additive sequence **cannot** have leading zeros, so sequence 1, 2, 03 or 1, 02, 3 is invalid.

**Example 1:**

Input: "112358"  
Output: true  
Explanation: The digits can form an additive sequence: 1, 1, 2, 3, 5, 8.   
 1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8

**Example 2:**

Input: "199100199"  
Output: true  
Explanation: The additive sequence is: 1, 99, 100, 199.   
 1 + 99 = 100, 99 + 100 = 199

**Constraints:**

* num consists only of digits '0'-'9'.
* 1 <= num.length <= 35

**Follow up:** How would you handle overflow for very large input integers?

**解**

* 回溯法。注意回溯中，在进入分支之后，还要退出分支
* 大数运算。用字符串实现大数的加减乘除

// 大数，只需要重载加法  
class BigNum{  
public:  
 string val;  
 BigNum(string s, bool hasreversed = false){  
 if(!hasreversed){  
 reverse(s.begin(), s.end());  
 }  
 val = s;  
 }  
 void Print(){  
 for(int i = val.size()-1; i>=0; --i)cout << val[i];  
 cout << endl;  
 }  
 //先加两个字符串公有的部分，再加上剩下的部分  
 //如果进位不为0，还要加上进位  
 BigNum operator + (BigNum &b){  
 string s1 = b.val, res;  
 int carry = 0, i = 0;  
 while(i < s1.size() && i < val.size()){  
 int tmp = carry + (s1[i]-'0') + (val[i]-'0');  
 carry = tmp / 10;  
 res += tmp % 10 + '0';  
 i++;  
 }  
 while(i < s1.size()){  
 int tmp = carry + (s1[i]-'0');  
 carry = tmp / 10;  
 res += tmp % 10 + '0';  
 i++;  
 }  
 while(i < val.size()){  
 int tmp = carry + (val[i]-'0');  
 carry = tmp / 10;  
 res += tmp % 10 + '0';  
 i++;  
 }  
 if(carry)res += carry + '0';  
 return BigNum(res, true);  
 }  
};  
  
class Solution {  
public:  
 bool isAdditiveNumber(string num) {  
 vector<BigNum>path;  
 return dfs(num, 0, path);  
 }  
 bool dfs(string &num, int pos, vector<BigNum>&path){  
 //结束条件，当pos到达了字符串末尾，则枚举完毕，看path的长度，如果大于2，说明前面的枚举是成功的  
 if(pos >= num.size())return path.size() >= 3;  
 int len = path.size();  
 // 如果len<2，说明最前面的两个数字还没有选出来，枚举选择即可  
 // 不能出现前缀0，即02 00这些是不合法的，只能选成0  
 if(len < 2){  
 for(int i = 1; i <= num.size(); ++i){  
 if(num[0] == '0' && i > 1)break;  
 path.push\_back(BigNum(num.substr(0, i)));  
 for(int j = i+1; j <= num.size(); ++j){  
 if(num[i] == '0' && j > i + 1)break;  
 path.push\_back(BigNum(num.substr(i, j-i)));  
 if(dfs(num, j, path)){  
 return true;  
 }else{  
 path.pop\_back();  
 }  
 }  
 path.pop\_back();  
 }  
 }else{  
 // 算出前两个数字的和，看对应的字符串是不是num的子串，不是的话返回，是的话往后搜索  
 BigNum next\_int = path[len-2] + path[len-1];  
 string next\_str = next\_int.val;  
 reverse(next\_str.begin(), next\_str.end());  
 if(num.substr(pos, next\_str.size()) == next\_str){  
 path.push\_back(next\_int);  
 bool res = dfs(num, pos + next\_str.size(), path);  
 path.pop\_back();  
 return res;  
 }else{  
 return false;  
 }  
 }  
 return false;  
 }  
};