# [306. 累加数]【字符串处理大整数加法】【回溯算法】

## 初始代码：缺陷（不能处理大整数的加法）

class Solution {  
 private:  
 //将字符转换为数字  
 int transform(int begin,int end,string&num){  
 int res=0;  
 //第一个数应该不会出现以0开头的情况  
 //但第二个数就不一定了  
 //0是可以存在的，但是以0开头的数是不可以存在的  
 //必须等于它之前两个数的和  
 //大整数相加会出现溢出的情况  
 //所以将整数相加转化为字符串相加  
 if(num[begin]=='0'&&begin!=end)return -1;  
 for(int i=begin;i<=end;i++){  
 res\*=10;  
 res+=num[i]-'0';  
 }  
 return res;  
 }  
  
 string str\_transform(int begin,int end,string&num){  
 if(num[begin]=='0'&&begin!=end)return "";  
 string res(num,begin,end-begin+1);  
 return res;  
 }  
  
 string add(string a, string b) {  
 string ans;  
 int len1 = a.size(), len2 = b.size(), len = min(len1, len2), i;  
 for (i = 0; i < len; i++) {  
 ans.push\_back(a[len1 - 1 - i] + b[len2 - 1 - i]-‘0’);//此处bug  
 }  
 if (len1 < len2) {  
 for (int j = len2 - len1 - 1; j >= 0; j--) {  
 ans.push\_back(b[j]);  
 }  
 }  
 else if (len1 > len2) {  
 for (int j = len1 - len2 - 1; j >= 0; j--) {  
 ans.push\_back(a[j]);  
 }  
 }  
  
 int car = 0,n=ans.size();  
 for (i = 0; i < n; i++) {  
 if (ans[i] + car >= '9' + 1) {  
 ans[i] = ans[i] + car - '9' - 1;  
 car = 1;  
 }  
 else {  
 ans[i] = ans[i] + car;  
 car = 0;  
 }  
 }  
 if (car == 1)ans.push\_back('1');//此处bug  
 string res;  
 for (i =n - 1; i >= 0; i--)res.push\_back(ans[i]);  
  
 return res;  
}  
  
  
 //回溯函数  
 //bool backtracking(int begin,string&num,vector<int>&myvec){  
 bool backtracking(int begin,string&num,vector<string>&myvec){  
 //终止条件  
 if(begin==num.size())return true;  
  
 for(int i=begin;i<num.size();i++){  
 //int temp=transform(begin,i,num);  
 //if(temp==-1)continue;  
 string temp=str\_transform(begin,i,num);  
 if(temp=="")continue;  
 //运用set函数来判断和查找两数之和  
 //根据题意，是它之前两个数的和，所以应该只用存储之前的两个数就可以了  
 //if(temp==myvec[0]+myvec[1]){  
 if(temp==add(myvec[0],myvec[1])){  
 myvec[0]=myvec[1];  
 myvec[1]=temp;  
 if(backtracking(i+1,num,myvec)){  
 return true;  
 }  
 break;  
 }  
 /\*  
 int flag=0;  
   
 unordered\_set<int>myset;  
 for(auto tmp:myvec){  
 auto iter=myset.find(temp-tmp);  
 if(iter!=myset.end()){  
 myvec.push\_back(temp);  
 flag=1;  
 break;  
 }  
 myset.insert(tmp);  
 }  
 if(flag){  
 if(backtracking(i+1,num,myvec)){  
 return true;  
 }  
 }\*/  
 }  
 return false;  
 }  
public:  
 bool isAdditiveNumber(string num) {  
 //寻找两个基数  
 //由于有符号和无符号的比较产生超出时间限制  
 int n=num.size();  
 for(int i=0;i<n-2;i++){  
 //此处i<num,size()-2;剪枝，因为至少需要三个数  
 //unordered\_set<int>myset;  
  
 //int num1=transform(0,i,num);  
 //if(num1==-1)continue;  
 string num1=str\_transform(0,i,num);  
 if(num1=="")continue;  
   
 for(int j=i+1;j<n-1;j++){  
 //int num2=transform(i+1,j,num);  
 //if(num2==-1)continue;  
 string num2=str\_transform(0,i,num);  
 if(num2=="")continue;  
 //设置一个集合并插入两个基数  
 //vector<int>myvec;  
 //myvec.push\_back(num1);  
 //myvec.push\_back(num2);  
 vector<string>myvec;  
 myvec.push\_back(num1);  
 myvec.push\_back(num2);  
  
 if(backtracking(j+1,num,myvec)){  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
};

## 处理大整数的加法：存在很多bug

class Solution {  
 private:  
 string str\_transform(int begin,int end,string&num){  
 if(num[begin]=='0'&&begin!=end)return "";  
 string res(num,begin,end-begin+1);  
 return res;  
 }  
   
 string add(string a, string b) {  
 string ans;  
 int len1 = a.size(), len2 = b.size(), len = min(len1, len2), i;  
 for (i = 0; i < len; i++) {  
 ans.push\_back(a[len1 - 1 - i] + b[len2 - 1 - i]);  
 }  
 if (len1 < len2) {  
 for (int j = len2 - len1 - 1; j >= 0; j--) {  
 ans.push\_back(b[j]);  
 }  
 }  
 else if (len1 > len2) {  
 for (int j = len1 - len2 - 1; j >= 0; j--) {  
 ans.push\_back(a[j]);  
 }  
 }  
  
 int car = 0,n=ans.size();  
 for (i = 0; i < n; i++) {  
 if (ans[i] + car >= '9' + 1) {  
 ans[i] = ans[i] + car - '9' - 1;  
 car = 1;  
 }  
 else {  
 ans[i] = ans[i] + car;  
 car = 0;  
 }  
 }  
 if (car == 1)ans.push\_back('1');  
 string res;  
 for (i =n - 1; i >= 0; i--)res.push\_back(ans[i]);  
  
 return res;  
}  
  
  
 //回溯函数  
 bool backtracking(int begin,string&num,vector<string>&myvec){  
 //终止条件  
 if(begin==num.size())return true;  
  
 for(int i=begin;i<num.size();i++){  
 string temp=str\_transform(begin,i,num);  
 if(temp=="")continue;  
   
 if(temp==add(myvec[0],myvec[1])){  
 myvec[0]=myvec[1];  
 myvec[1]=temp;  
 if(backtracking(i+1,num,myvec)){  
 return true;  
 }  
 break;  
 }  
 }  
 return false;  
 }  
public:  
 bool isAdditiveNumber(string num) {  
   
 //由于有符号和无符号的比较产生超出时间限制  
 int n=num.size();  
 for(int i=0;i<n-2;i++){  
   
 string num1=str\_transform(0,i,num);  
 if(num1=="")continue;  
   
 for(int j=i+1;j<n-1;j++){  
   
 string num2=str\_transform(0,i,num);  
 if(num2=="")continue;  
   
 vector<string>myvec;  
 myvec.push\_back(num1);  
 myvec.push\_back(num2);  
  
 if(backtracking(j+1,num,myvec)){  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
};

## 完善代码

#include<iostream>  
#include<string>  
#include<vector>  
using namespace std;  
  
  
class Solution {  
private:  
 string str\_transform(int begin, int end, string& num) {  
 if (num[begin] == '0' && begin != end)return "";  
 string res(num, begin, end - begin+1 );  
 return res;  
 }  
  
 string add(string a, string b) {  
 string ans;  
 int len1 = a.size(), len2 = b.size(), len = min(len1, len2), i;  
 for (i = 0; i < len; i++) {  
 ans.push\_back(a[len1 - 1 - i] + b[len2 - 1 - i]-'0');  
 }  
 if (len1 < len2) {  
 for (int j = len2 - len1 - 1; j >= 0; j--) {  
 ans.push\_back(b[j]);  
 }  
 }  
 else if (len1 > len2) {  
 for (int j = len1 - len2 - 1; j >= 0; j--) {  
 ans.push\_back(a[j]);  
 }  
 }  
  
 int car = 0, n = ans.size();  
 for (i = 0; i < n; i++) {  
 if (ans[i] + car >= '9' + 1) {  
 ans[i] = ans[i] + car - '9' - 1+'0';  
 car = 1;  
 }  
 else {  
 ans[i] = ans[i] + car;  
 car = 0;  
 }  
 }  
 if (car == 1) {  
 ans.push\_back('1');  
 n++;  
 }  
 string res;  
 for (i = n - 1; i >= 0; i--)res.push\_back(ans[i]);  
  
 return res;  
 }  
  
  
 //回溯函数  
 bool backtracking(int begin, string& num, vector<string>& myvec) {  
 //终止条件  
 if (begin == num.size())return true;  
  
 for (int i = begin; i < num.size(); i++) {  
 string temp = str\_transform(begin, i, num);  
 if (temp == "")continue;  
  
 cout << "temp:" << temp << endl;  
 cout << "add:" << add(myvec[0], myvec[1]) << endl;  
 if (temp == add(myvec[0], myvec[1])) {  
 myvec[0] = myvec[1];  
 myvec[1] = temp;  
 if (backtracking(i + 1, num, myvec)) {  
 return true;  
 }  
 break;  
 }  
 }  
 return false;  
 }  
public:  
 bool isAdditiveNumber(string num) {  
  
 //由于有符号和无符号的比较产生超出时间限制  
 int n = num.size();  
 for (int i = 0; i < n - 2; i++) {  
  
 string num1 = str\_transform(0, i, num);  
 if (num1 == "")continue;  
 cout << "num1:" << num1 << endl;  
  
 for (int j = i + 1; j < n - 1; j++) {  
  
 string num2 = str\_transform(i+1, j, num);  
 if (num2 == "")continue;  
  
 cout << "num2:" << num2 << endl;  
  
 vector<string>myvec;  
 myvec.push\_back(num1);  
 myvec.push\_back(num2);  
  
 if (backtracking(j + 1, num, myvec)) {  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
};  
string add1(string a, string b) {  
 string ans;  
 int len1 = a.size(), len2 = b.size(), len = min(len1, len2), i;  
 for (i = 0; i < len; i++) {  
 ans.push\_back(a[len1 - 1 - i] + b[len2 - 1 - i]-'0');  
 }  
 if (len1 < len2) {  
 for (int j = len2 - len1 - 1; j >= 0; j--) {  
 ans.push\_back(b[j]);  
 }  
 }  
 else if (len1 > len2) {  
 for (int j = len1 - len2 - 1; j >= 0; j--) {  
 ans.push\_back(a[j]);  
 }  
 }  
  
 int car = 0, n = ans.size();  
 for (i = 0; i < n; i++) {  
 if (ans[i] + car >= '9' + 1) {  
 ans[i] = ans[i] + car - '9' - 1+'0';  
 car = 1;  
 }  
 else {  
 ans[i] = ans[i] + car;  
 car = 0;  
 }  
 }  
 if (car == 1) {  
 ans.push\_back('1');  
 n++ ;  
 }  
 string res;  
 for (i = n - 1; i >= 0; i--)res.push\_back(ans[i]);  
  
 return res;  
}  
int main() {  
 Solution A;  
 string num = "199100199";  
 cout<<A.isAdditiveNumber(num)<<endl;  
 string s1 = "1980",s2="1982";  
 string res = add1(s1, s2);  
 cout << res << endl;  
 string s3 = s1 + s2;  
 cout << s3 << endl;  
 s1 = s2;  
 cout << s2 << endl;  
 return 0;  
   
}