信息学奥赛笔记12

高精度算法（CSP-J 赛大概率考！！！ 重要！！）

一、上节课课后作业

[B2094]不与最大数相同的数字之和https://www.luogu.com.cn/problem/B2094

题目描述

输出一个整数数列中不与最大数相同的数字之和。

输入格式

输入分为两行：

第一行为 N（N 为接下来数的个数，$N \leq 100$）；

第二行为 N 个整数，数与数之间以一个空格分开，每个整数的范围是 $-1000,000$ 到 $1000,000$。

输出格式

输出为 N 个数中除去最大数其余数字之和。

样例 #1

样例输入 #1

```
3
1 2 3
```

样例输出 #1

```
3
```

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
int n, m;
long long ans;
int main() {
    cin >> n;
    vector<int> a;
    for (int i = 0; i < n; i++){
```

```
        cin >> x;
        a.push_back(x);
        m = max(m, x);
    }
    for (int i = 0; i < n; i++) {
        if (x != m) {
            ans += x;
        }
    }
    cout << ans <<endl;
    return 0;
}
```

[B2096]直方图 https://www.luogu.com.cn/problem/B2096

题目描述

给定一个非负整数数组，统计里面每一个数的出现次数。我们只统计到数组里最大的数。

假设 $Fmax$（$Fmax \leq 100000$）是数组里最大的数，那么我们只统计 $\{0, 1, 2 \dots Fmax\}$ 里每个数出现的次数。

输入格式

第一行 n 是数组的大小。$1 \leq n \leq 100000$。

紧接着一行是数组的 n 个元素。

输出格式

按顺序输出每个数的出现次数，一行一个数。如果没有出现过，则输出 0。

对于例子中的数组，最大的数是 3，因此我们只统计 $\{0, 1, 2, 3\}$ 的出现频数。

样例 #1

样例输入 #1

```
5
1 1 2 3 1
```

样例输出 #1

```
0
3
1
1
```

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
int x, n, m;
int main() {
    cin >> n;
    vector<int> a(100001);
    for (int i = 0; i < n; i++) {
        cin >> x;
        a[x]++;
        m = max(m, x);
    }
    for (int i = 0; i<= m; i++) {
        cout << a[i] << endl;
    }
    return 0;
}
```

[B2064]斐波那契数列https://www.luogu.com.cn/problem/B2064

题目描述

斐波那契数列是指这样的数列：数列的第一个和第二个数都为 1，接下来每个数都等于前面 2 个数之和。

给出一个正整数 a，要求斐波那契数列中第 a 个数是多少。

输入格式

第 1 行是测试数据的组数 n，后面跟着 n 行输入。每组测试数据占 1 行，包括一个正整数 a（$1 \leq a \leq 30$）。

输出格式

输出有 n 行，每行输出对应一个输入。输出应是一个正整数，为斐波那契数列中第 a 个数的大小。

样例 #1

样例输入 #1

```
4
5
2
19
1
```

样例输出 #1

```
5
1
4181
1
```

```cpp
#include <iostream>
#include <vector>
using namespace std;
int t, x, i;
int main() {
    vector<int> a = {0, 1, 1}; //初始化斐波那契数列的前两位，由于vector数组从0开始，我们多添加一个数，把答案对齐到a[1], a[2];
    cin >> t;
    for (int i = 0; i < t; i++) { //一共需要输入t组数据
        cin >> x;  //输出斐波那契数列的a[x];
        i = a.size();  //获取当前求出的斐波那契数列的长度
        if (x >= i) {  //假如要求的a[x]大于我们目前已经求了的长度，需要拓展目前的动态数组
            for (; i <= x; i++) { //一直求答案求到a[x]为止
                a.push_back(a[i - 1] + a[i - 2]);
            }
        }//假如要求的a[x]小于目前的数组长度，说明咱们是不是就可以直接输出了，因为a[x]的值已经有了
        cout << a[x] << endl;
    }
    return 0;
}
```

二、本节课新知识——高精度算法

> 在题目中，有些时候需要计算的数据已经超过了long long 的最大值(9223372036854775807)，比如100000000000000000 * 1000000000000000000；
>
> 在这些很大数字的运算中，我们想要去运算结果，该如何去做呢？大家在学习数学的过程中是不是接触到了竖式加法，减法，乘法和除法，我们是不是可以仿照数学里的列竖式的形式，来操作待计算的数，这样我们就可以对很大的数字也能获取到运算结果了。

本节课中，我们主要学习了**高精度加法**

$$\begin{array}{r} 1234 \\ +559 \\ \hline 1793 \end{array}$$

像这种列竖式型的加法运算不就可以解决数据极大无法通过C++给我们提供的标准数据存储格式int或long long的限定了吗

那我们是不是可以使用$string$来代替数字存储呀，那么大家可以发现，字符串里面存储的是字符，我们没法直接对字符进行加减法运算，比如$'0'+'1'$结果是第97号字符$'a'$（因为$48+49=97$），所以，咱们可以先把字符串拆成一个个整数，放进数组里，这就涉及到了字符转数字的做法；

v1数组下标i	0	1	2	3
数组里的值	1	2	3	4

v2数组下标j	0	1	2	
数组里的值	5	5	9	

在加法中，不仅仅是两个数直接相加，还涉及到了进位问题，比如$9+5=14$在结果中，我们应该保留个位的4，舍去十位的1并放置到进位格上，所以，咱们需要一个变量a帮助我们来统计当前的数是否需要进位，那么对于一个位置i来说，他的结果真正的计算方法应该是

$$v1[i]+v1[i]+v2[i]+a$$

既然我们做加法的时候是从右往左的，所以咱们在运算的时候也应该从右往左运算，各位同学可以发现我们$v1[0]+v2[0]$ 是不是不太对，它并不是最右侧两个数相加。所以我们要把两个数组全部倒过来运算，这里就需要使用到C++为我们提供的内置函数$reverse$，使用这个函数需要调用头文件$\#include<algorithm>$。用法如下：

```
vector<int> vec;
reverse(vec.begin(), vec.end());

reverse(数组名.begin(), 数组名.end());
```

那么，我们就获得了两个新的数组

v1数组下标i	0	1	2	3
数组里的值	4	3	2	1

v2数组下标j	0	1	2	
数组里的值	9	5	5	

现在我们就可以直接对这两个数组做加法运算了，那么为了节省空间，我们是不是可以把答案保存到那个较长的加数数组里，因为两个数相加，结果的长度一定是$\geq$这两个加数中位数较多的那个数的位数。所以当$v1$的长度短于$v2$时，咱们是不是可以交换两个数组的值，这里我们又要介绍一下<algorithm>当中的另外一个函数$swap$。

```
假设a和b是两个相同类型的变量
swap(a, b)即可交换两个变量的值

int a = 5, b = 7;
swap(a, b);
此时 a = 7, b = 5;

```

```
同理，我们也可以交换两个数组的值
vector<int> a = {1, 2, 3, 4};
vector<int> b = {5, 6, 7};
swap(a, b);
现在a = {5, 6, 7}, b = {1, 2, 3, 4};

但是以下这种情况是不被允许的
int a = 5;
long long b = 1234567890;
swap(a, b);
因为a, b是两个不同类型的变量，所以无法完成交换。

**********拓展部分**********:
那咱们该如何临时的把int转成long long运算呢？
C++为我们提供了另外一个用法
1LL的值就是1，但是不同的是，计算机不会把这个1LL认为是整数int型的1，
而是会认为它是long long类型的1
所以我们可以通过把一个整数int 乘以 1LL来临时的将他转变为long long;
int a = 5;
long long b = 7, c;
c = max(a, b);
这个代码计算机就会报错，因为a, b不同类型。但是咱们可以这么修改--
int a = 5;
long long b = 7, c;
c = max(1LL * a, b);
这样就不会报错了，因为在这里和b比较大小的是a * 1LL这个整体，他会临时的被计算机认为是个long long类型
这样我们就可以不转换a变量的数据类型，直接让它参与long long运算了
```

通过交换$v1$和$v2$的值，我们就可以保证$v1.size() \geq v2.size()$。

那我们进一步思考，发现如果$v2$的所有加数都被加完了，是不是就代表我们的加法做完了呢？还没有，因为$v1$可能此时仍然在使用$v2$的进位继续运算，所以我们加完了$v2$后，仍然要把$v1$遍历完才可以。代码做到这就算结束了吗？

同学们可以思考一下这个样例

> 999 + 1 = 1000

在这个样例中，如果我们按照一位一位的处理，是不是发现$v1$最终保存的答案是多少？是不是000，为什么？因为最后一次还有一个进位咱们没有处理，如果当长的那个加数已经遍历完仍然发现进位数还有的话，说明我们的答案需要有一个最高位的1，此时我们就可以使用上节课说到的$vector$中的$push_back$功能，来给答案最后直接补1就行，做到这里，代码该结束了吧？

还没有！！

我们来看看刚刚那个例子的结果。

v1数组下标i	**0**	**1**	**2**	**3**
数组里的值	3	9	7	1

因为咱们是倒着处理的，这样就导致我们的答案也是倒着的，所以我们最后的最后，是不是还需要把答案给他颠倒回来。所以咱们还需要再调用一次$reverse$函数！这一步至关重要！

这样我们就可以写出代码了。

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
vector<int> string_to_int(string s) {//这个函数的作用是，将字符串的每一位拆开放进数组里
    vector<int> v;
    for(int i = 0; i < s.size(); i++)   {
        int n = s[i] - '0';//字符数字 - '0' = 对应的整数数字
        v.push_back(n);
    }
    return v;
}
vector<int> add(vector<int> v1, vector<int> v2) {
    if(v1.size() < v2.size()) { //v1需要用来保存答案，如果v1的长度比v2小，需要交换两个数组的值
        swap(v1,v2);
    }
    reverse(v1.begin(), v1.end());//对v1和v2做反转
    reverse(v2.begin(), v2.end());
    int a = 0;
    for(int i = 0; i < v2.size(); i++) {
        v1[i] = v1[i] + v2[i] + a;
        if(v1[i]>9) { //如果当前的结果大于9，说明需要进位了
            v1[i] -= 10; //先把结果 - 10，这里最好是 % 10,原因我们在下面的优化中说
            a = 1; //进位数计为1
        } else {
            a = 0;//如果当前位的结果小于10，说明不需要进位，把进位值设为0
        }
    }
    for(int i = v2.size(); i < v1.size(); i++) {//遍历完v2后，此时我们要继续扫描v1，防止还有进位没有处理
        v1[i] = v1[i] + a;//直接就是v1[i] + a
        if(v1[i] > 9) {
            v1[i] -= 10;
            a = 1;
        } else {
            a = 0;
        }
    }
    if(a == 1) {//如果等v1扫描完了还发现有进位，说明答案需要补最高位
        v1.push_back(1);
    }
    reverse(v1.begin(), v1.end());//答案是反着的，我们要把答案颠倒回来
    return v1;
}
int main() {
    string s1, s2;
    cin >> s1 >> s2;
    vector<int> v1, v2;
    v1 = string_to_int(s1);
    v2 = string_to_int(s2);
    vector<int> v3 = add(v1, v2);
    for(int i = 0; i < v3.size(); i++) {
        cout << v3[i];
    }
```

```
    return 0;
}
```

优化

> 那么同学们有没有发现，我们先把两个加数都倒过来，最后在求答案的时候又倒回去了，在这方面，我们是不是可以考虑一下优化呢?
>
> 把两个加数全都倒过来的好处是什么？是不是我们可以从数组的0位开始同时遍历，方便我们从左往右加，和正常思维过程一样，那么请大家转变一下思维，假如要是倒过来做呢？像我们真正的列竖式那样，先处理最右边的个位，再处理最右边的十位.....该怎么做呢。
>
> 在这里我们需要使用到一个思想——**双指针(Double Pointers)**，咱们可以用一个变量i指向第一个加数$v1$的末尾，用一个变量j指向第二个加数$v2$的末尾，当$i \geq 0$是不是就说明$v1$仍然没有加完，还有数可加，当$j \geq 0$是不是就说明$v2$还没加完，那么大家想一想，大部分情况下，两个加数的位数都是不同的，万一有一个加数优先被加完了呢?
>
> 在第一种方法中，我们是如何解决的，通过两次遍历，第一次把第二个短的加数都加一遍，第二次处理进位符号，那么我们还可以怎么做呢?
>
> 如果我们只考虑上一个数加下一个数，咱们是不是可以把第二个加数中已经超过加数的部分置为0
>
> 例如下面:

$$\begin{array}{r} 1234 \\ +0559 \\ \hline 1793 \end{array}$$

> 这样的话，我们就可以直接对原始数据从右往左运算了，咱们用a来代表$v1$中当前的加数$v1[i]$，用b来代替$v2$中的当前加数$v2[j]$，用x来表示进位，那么对于任何一位的结果，我们是不是都可以表示成$(a+b+x)\%10$，比如例子中的个位数应该是$(4+9+0)\%10=3$，所以答案的最后一位写3
>
> 对任何一位的进位，我们可以表示成$(a+b+x)/10$，比如例子中的最后一位，$(4+9+0)/10=1$，接下来的十位需要进1位。
>
> 那如果对于$i<0$是不是就说明$v1$已经被加完了，我们就可以把a置为0。
>
> 如果对于$j<0$就说明$v2$已经加完了，那么我们把b置为0，这样的话，咱们就不需要进行2次遍历，一次就可以搞定，也不需要先把$v1$和$v2$倒过来处理了，直接逆序遍历。那么，大家可以想一想，什么时候循环才会停止呢，是不是在$i \geq 0 \quad || \quad j \geq 0$的时候都表示仍然有加数没处理完，继续循环，所以咱们可以改写成$while$循环。

```
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>
using namespace std;
vector<int> string_to_int(string s) {
    vector<int> v;
```

```
    for(int i = 0; i < s.size(); i++) {
        v.push_back(s[i] - '0');
    }
    return v;
}
vector<int> add(vector<int> v1, vector<int> v2) {
    vector<int> v3;
    int i = v1.size() - 1, j = v2.size() - 1, x = 0; //i从v1末尾开始，j从v2末尾开始，x表示进位
    while(i >= 0 || j >= 0) { //只要有任何一个数还没加完，循环都继续进行
        int a = 0, b = 0; //用a保存第一个加数，b保存第二个加数，他们默认为0
        if(i >= 0) {  //如果第一个加数还没完
            a = v1[i--]; //a保存第一个加数，并且i往前移动一个
        }
        if(j >= 0) {  //如果第二个加数还没完
            b = v2[j--]; //b保存第二个加数，并且j往前移动一个
        }
        x += a + b;  // x = x + a + b，我们把结果先保存进进位变量x中
        v3.push_back(x % 10); //结果应该添加x % 10;
        x /= 10;     //进位数 = x / 10;

    }
    if(x == 1) {  //如果处理完所有的加数，发现进位数仍然不为0
        v3.push_back(1); //说明需要补上一个1
    }
    reverse(v3.begin(),v3.end());  //计算是从右往左的，所以答案每次添加进来是反的，最终需要反转
    return v3;
}
int main() {
    string s1, s2;
    cin >> s1 >> s2;
    vector<int> v1, v2;
    v1 = string_to_int(s1);
    v2 = string_to_int(s2);
    vector<int> v3 = add(v1, v2);
    for(int i = 0; i < v3.size(); i++) {
        cout<<v3[i];
    }
    return 0;
}
```

进一步优化

> 对于原始字符串来说，其实咱们也没有必要把他们先转成vector处理，咱们可以直接从string上下手！
>
> 终极高精度加法代码！！
>
> 以下代码需要同学们**完全掌握理解并记忆**

```
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>
```

```
using namespace std;
string add(string s1, string s2) { //传入的值直接变成了string，不需要将他先转变成数组vector
    string ans;//所以答案也直接是string
    int i = s1.size() - 1, j = s2.size() - 1, x = 0;
    while(i >= 0 || j >= 0) {
        int a = 0, b = 0;
        if(i >= 0) {
            a = s1[i--] - '0'; //由于a是int，所以我们在这里才把v1[i]转变为int类型
        }
        if(j >= 0) {
            b = s2[j--] - '0';
        }
        x += a + b;
        ans += x % 10 + '0';
        //由于ans是string类型，所以我们需要把int类型的x % 10重新变为char类型，string类型可以通过自增符号来push_back
        x /= 10;

    }
    if (x) {
        ans += "1"; //如果最终还有进位符，我们可以直接在答案字符串的末尾添加字符串"1"
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
int main() {
    string s1, s2;
    cin >> s1 >> s2;
    cout << add(s1, s2) << endl;
    return 0;
}
```

三目运算符优化

> 再进一步的，我们可以通过三目运算符来继续优化代码的长短，方便各位同学们记忆并使用。

```
如果一个变量的赋值不是x则是y我们可以这么写
int a;
if (条件成立) {
    a = x;
} else {
    a = y;
}
以上代码一共是5行，在C++中，有一种运算规则叫做三目运算符，它为我们提供了非常简短的写法，比如刚刚的例子
int a;
if (a % 2 == 1) {
    a = 119;
} else {
    a = 911;
}
我们可以用三目运算符优化为
int a = a % 2 == 1 ? 119 : 911;
三目运算符可以这么拆解：
```

```
a = 一个数，这个数有条件
a % 2 == 1 成立吗？
如果成立的话a = 119，否则 a = 911;
int a = （条件）? 成立值 : 不成立值
```

高精度加法最优代码：

```cpp
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>
using namespace std;
string add(string s1, string s2) {
    string ans;
    int i = s1.size() - 1, j = s2.size() - 1, x = 0;
    while(i >= 0 || j >= 0) {
        int a = i >= 0 ? s1[i--] - '0' : 0; //这里对a使用了三目运算符
        int b = j >= 0 ? s2[j--] - '0' : 0; //这里对b使用了三目运算符
        x += a + b;
        ans += x % 10 + '0';
        x /= 10;
    }
    if (x) ans += "1";
    reverse(ans.begin(),ans.end());
    return ans;
}
int main() {
    string s1, s2;
    cin >> s1 >> s2;
    cout << add(s1, s2) << endl;
    return 0;
}
```

以上代码中， 6-19行是同学们需要最终背诵并能熟练地默写的代码，这个函数的作用就是丢进去两个极大的字符串数字，传出他们的和的字符串，在考试中如果需要进行大数运算的时候，原本的a + b就可以改写成c = add(a, b)的形式，这样我们就不受限于long long也不够大了！

你学会了吗？

背代码！！！！！！！！！