C++方向编程题答案

第六周

day33

1、题目ID：837 剪花布条

链接：https://www.nowcoder.com/questionTerminal/1046cd038f7a4b04b2b77b415973de1c

【题目解析】：

题目简单描述下，就是在S串中，T串整体出现了多少次。

【解题思路】：

C语言可以通过strstr函数找，用STL的string库可以通过find函数找，找到以后跳过一个T串的长度。例如：在abcacbcbcabscbc中找cbc，第一次找到了这个位置：abca**cbcbcabscbc**，找到这个下标后，会跳过整体cbc，也就是从这个位置继续找：abcacbc**bcabscbc**，否则如果你只跳一个字符，会导致cbcbc会被算成2次，而按照本题的题意，应该算一次。

【示例代码】：

```
#include<iostream>
#include<string>


int main()
{
    std::string s, t;

    while (std::cin >> s >> t)
    {
        int res = 0;
        size_t pos = 0;
        //依次在 s 中查找 t 即可，直到再也找不到 t
        while ((pos = s.find(t, pos)) != std::string::npos)
        {
            pos += t.size(); //跳过t串整体的长度
            ++res; //计数
        }

        std::cout << res << std::endl;
    }

    return 0;
}
```

2、题目ID：749 客似云来

链接：https://www.nowcoder.com/questionTerminal/3549ff22ae2c4da4890e9ad0ccb6150d

【题目解析】：

题目其实是让你求出斐波那契数列中某一段的和。

【解题思路】：

老样子，先准备好斐波那契的数组，然后遍历那一段数组，求出他们的和即可。而第80项斐波那契数列是一个17位数，所以需要用long long来解决问题。

然而这个题还有另一个更有意思的思路。斐波那契数列的的前n项和其实是有一个很有意思的公式，公式推导在这里https://blog.csdn.net/ftx456789/article/details/82348742，根据文章我们能知道，斐波那契数列的前n项和，就是第n+2项的值减1，例如前10项的和143，就是第12项的144 - 1的结果。所以，我们如果我们要第n项到第m项的和，那么只要求出前m项的和，减去前n - 1项的和，就能得到结果了。例如要求第3项到第5项的和，我们就只需要用前5项的和减去前2项的和，而公式中的减一在这个过程中抵消掉了，也就是结果直接就是第7项的值减去第4项的值，这样我们在操作的时候就更简单了。就数值而言，第7项是13，第4项是3，差值是10，而2+3+5也是10，结果是正确的。

【示例代码】：

```
#include <iostream>
#define MAX 83 //如果用公式计算，需要接下来两项的值

void solve(long long num[])
{
    for (int i = 2; i < MAX; i++)
    {
        num[i] = num[i - 1] + num[i - 2];
    }
}

long long sum_traversal(long long num[], int from, int to) //解法1：用遍历求和求解
{
    long long ans = 0;
    for (int i = from - 1; i < to; i++) //让数组下标从from - 1遍历到to - 1
    {
        ans += num[i];
    }
    return ans;
}

long long sum_formula(long long num[], int from, int to) //解法2：用公式求解
{
    return num[to + 1] - num[from]; //第to + 2项的下标是to + 1，第from + 2 - 1项的下标是from
}

int main()
{
    int from, to;
    long long num[MAX] = {1, 1};
    //提前计算Fibonacci数列
    solve(num);
    while (std::cin >> from >> to)
    {
        std::cout << sum_formula(num, from, to) << std::endl; //两个方法二选一。公式更快。
    }
    return 0;
}
```