考核讲评 + 递归初步

第二堂课

Garen-Wang

2018 年 6 月 24 日

目录

课前废话

课前废话

这个学期的第一次考核在 06.05 举行。

课前废话

这个学期的第一次考核在 06.05 举行。
我题目的难度难道打反了吗？T2 难道比 T1 难吗？

课前废话

这个学期的第一次考核在 06.05 举行。
我题目的难度难道打反了吗？T2 难道比 T1 难吗？
目前最多 100 分（除了我啊）。实力还有待提高哦！

课前废话

这个学期的第一次考核在 06.05 举行。
我题目的难度难道打反了吗？T2 难道比 T1 难吗？
目前最多 100 分（除了我啊）。实力还有待提高哦！
赛后精简了许多划水的成员，期末考后是真正的优胜劣汰了。（可怕）

课前废话

这个学期的第一次考核在 06.05 举行。
我题目的难度难道打反了吗？T2 难道比 T1 难吗？
目前最多 100 分（除了我啊）。实力还有待提高哦！
赛后精简了许多划水的成员，期末考后是真正的优胜劣汰了。（可怕）
课件好像有点多，我慢下来的时候记得提醒我开 1.5 倍速哦！

P1079 Vigenère 密码

P1079 Vigenère 密码

这道题最大的问题应该在于这张表。

P1079 Vigenère 密码

这道题最大的问题应该在于这张表。
可以发现这张表有一种规律：

P1079 Vigenère 密码

这道题最大的问题应该在于这张表。
可以发现这张表有一种规律：
设第一个字母（密钥）为 α，第二个字母（明文）为 β，则密文为：

$$\alpha + \beta -' A'$$

P1079 Vigenère 密码

这道题最大的问题应该在于这张表。
可以发现这张表有一种规律：
设第一个字母（密钥）为 α，第二个字母（明文）为 β，则密文为：

$$\alpha + \beta -' A'$$

所以明文的计算公式只需要移项就知道了。

P1079 Vigenère 密码

这道题最大的问题应该在于这张表。
可以发现这张表有一种规律：
设第一个字母（密钥）为 α，第二个字母（明文）为 β，则密文为：

$$\alpha + \beta -' A'$$

所以明文的计算公式只需要移项就知道了。
但是可能会超出范围诶！

P1079 Vigenère 密码

这道题最大的问题应该在于这张表。
可以发现这张表有一种规律：
设第一个字母（密钥）为 α，第二个字母（明文）为 β，则密文为：

$$\alpha + \beta -' A'$$

所以明文的计算公式只需要移项就知道了。
但是可能会超出范围诶！
如果从下超出范围，就加 26。如果从上超出范围，就减 26。

P1079 Vigenère 密码

这道题最大的问题应该在于这张表。
可以发现这张表有一种规律：
设第一个字母（密钥）为 α，第二个字母（明文）为 β，则密文为：

$$\alpha + \beta -' A'$$

所以明文的计算公式只需要移项就知道了。
但是可能会超出范围诶！
如果从下超出范围，就加 26。如果从上超出范围，就减 26。
所以密码表就这么解决了！

P1079 Vigenère 密码

P1079 Vigenère 密码

题目要求运算忽略大小写，保持原来的大小写形式。

P1079 Vigenère 密码

题目要求运算忽略大小写，保持原来的大小写形式。
那就记录下原来是大写还是小写，运算时全部转为大写或者小写，输出就按原样输出。

P1079 Vigenère 密码

题目要求运算忽略大小写，保持原来的大小写形式。
那就记录下原来是大写还是小写，运算时全部转为大写或者小写，输出就按原样输出。
我挑这道题的时候是一遍过的，所以把他放 T1 了。

P1079 Vigenère 密码

题目要求运算忽略大小写，保持原来的大小写形式。
那就记录下原来是大写还是小写，运算时全部转为大写或者小写，输出就按原样输出。
我挑这道题的时候是一遍过的，所以把他放 T1 了。
我事后还写了一波解题报告，搜我 luogu 的 blog 就有了。

P1098 字符串的展开

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。
想要展开必须同时满足三个条件：

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。
想要展开必须同时满足三个条件：

- 出现了减号
- 减号两侧同为小写字母或同为数字
- 减号右边的字符严格大于左边的字符

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。
想要展开必须同时满足三个条件：

- 出现了减号
- 减号两侧同为小写字母或同为数字
- 减号右边的字符严格大于左边的字符

对 p_1 来说，当 1 时保证展开的字母是小写的，所以当原来字母是大写的时候加上 32。

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。
想要展开必须同时满足三个条件：

- 出现了减号
- 减号两侧同为小写字母或同为数字
- 减号右边的字符严格大于左边的字符

对 p_1 来说，当 1 时保证展开的字母是小写的，所以当原来字母是大写的时候加上 32。
当 2 时也同理，减掉 32，当 3 时就无脑输出 p_2 个星号即可。

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。
想要展开必须同时满足三个条件：

- 出现了减号
- 减号两侧同为小写字母或同为数字
- 减号右边的字符严格大于左边的字符

对 p_1 来说，当 1 时保证展开的字母是小写的，所以当原来字母是大写的时候加上 32。
当 2 时也同理，减掉 32，当 3 时就无脑输出 p_2 个星号即可。
对 p_3，正序就正向 for 循环，逆序就逆向 for 循环。

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。
想要展开必须同时满足三个条件：

- 出现了减号
- 减号两侧同为小写字母或同为数字
- 减号右边的字符严格大于左边的字符

对 p_1 来说，当 1 时保证展开的字母是小写的，所以当原来字母是大写的时候加上 32。
当 2 时也同理，减掉 32，当 3 时就无脑输出 p_2 个星号即可。
对 p_3，正序就正向 for 循环，逆序就逆向 for 循环。
减号两边字母一定不受展开的影响，所以照输即可。

P1098 字符串的展开

这道题好像会的人挺多，就随便讲讲了。
暴力模拟，题目的意思已经很清楚了。
想要展开必须同时满足三个条件：

- 出现了减号
- 减号两侧同为小写字母或同为数字
- 减号右边的字符严格大于左边的字符

对 p_1 来说，当 1 时保证展开的字母是小写的，所以当原来字母是大写的时候加上 32。
当 2 时也同理，减掉 32，当 3 时就无脑输出 p_2 个星号即可。
对 p_3，正序就正向 for 循环，逆序就逆向 for 循环。
减号两边字母一定不受展开的影响，所以照输即可。
我同样也写了解题报告。

这里是分割线！

跳出 main 函数！

跳出 main 函数！

在我们初步的理解中，我们知道 main 函数就是程序的开始，从 main 的前面开始，慢慢地走到最后的 return 0 结束。

跳出 main 函数！

在我们初步的理解中，我们知道 main 函数就是程序的开始，从 main 的前面开始，慢慢地走到最后的 return 0 结束。
但是只局限于 main 函数之中是有坏处的。For example:

跳出 main 函数！

在我们初步的理解中，我们知道 main 函数就是程序的开始，从 main 的前面开始，慢慢地走到最后的 return 0 结束。
但是只局限于 main 函数之中是有坏处的。For example:

- 数组不能开太大（例如 1000005），不然爆栈。
- 每次定义变量总少不了 memset 或者初始化。
- 有时候有重复性语言，通过复制粘贴很冗余，还容易错。

跳出 main 函数！

在我们初步的理解中，我们知道 main 函数就是程序的开始，从 main 的前面开始，慢慢地走到最后的 return 0 结束。
但是只局限于 main 函数之中是有坏处的。For example:

- 数组不能开太大（例如 1000005），不然爆栈。
- 每次定义变量总少不了 memset 或者初始化。
- 有时候有重复性语言，通过复制粘贴很冗余，还容易错。

所以跟着本人的脚步，在 main 函数的前面打上一个空格，这是新的开始！

全局变量与函数

全局变量与函数

在（main）函数外定义的变量、数组等叫做全局变量，在函数内定义的就是局部变量。

全局变量与函数

在（main）函数外定义的变量、数组等叫做全局变量，在函数内定义的就是局部变量。
定义全局变量有个优点：变量值自动置为 0！同时空间可开大很多！

全局变量与函数

在（main）函数外定义的变量、数组等叫做全局变量，在函数内定义的就是局部变量。
定义全局变量有个优点：变量值自动置为 0！同时空间可开大很多！
函数的一个优点就是面向对象语言中所谓的“封装”。

全局变量与函数

在（main）函数外定义的变量、数组等叫做全局变量，在函数内定义的就是局部变量。
定义全局变量有个优点：变量值自动置为 0！同时空间可开大很多！
函数的一个优点就是面向对象语言中所谓的“封装”。
将一个功能写进一个函数里面，使这个函数实现相应的功能，而可以在其他函数（甚至自己）中可以调用它。

全局变量与函数

在（main）函数外定义的变量、数组等叫做全局变量，在函数内定义的就是局部变量。
定义全局变量有个优点：变量值自动置为 0！同时空间可开大很多！
函数的一个优点就是面向对象语言中所谓的“封装”。
将一个功能写进一个函数里面，使这个函数实现相应的功能，而可以在其他函数（甚至自己）中可以调用它。
让我小小地举一个例子。比如老掉牙的判断质数。

全局变量与函数

在（main）函数外定义的变量、数组等叫做全局变量，在函数内定义的就是局部变量。
定义全局变量有个优点：变量值自动置为 0！同时空间可开大很多！
函数的一个优点就是面向对象语言中所谓的“封装”。
将一个功能写进一个函数里面，使这个函数实现相应的功能，而可以在其他函数（甚至自己）中可以调用它。
让我小小地举一个例子。比如老掉牙的判断质数。
而在一个函数中调用自己，就叫做递归 (recursion)。

认识递归

认识递归

首先让大家感性地认识递归。

认识递归

首先让大家感性地认识递归。
大家看网络直播的时候有没有见过主播的直播软件屏幕中出现了主播自己的直播软件屏幕？

认识递归

首先让大家感性地认识递归。
大家看网络直播的时候有没有见过主播的直播软件屏幕中出现了主播自己的直播软件屏幕？
这可以看作是一种递归！随着时间的增长，从屏幕中看到的主播人头会越来越多！

认识递归

首先让大家感性地认识递归。
大家看网络直播的时候有没有见过主播的直播软件屏幕中出现了主播自己的直播软件屏幕？
这可以看作是一种递归！随着时间的增长，从屏幕中看到的主播人头会越来越多！
如果主播不最小化他的直播软件屏幕，那么这种递归会无休止地进行！

认识递归

首先让大家感性地认识递归。
大家看网络直播的时候有没有见过主播的直播软件屏幕中出现了主播自己的直播软件屏幕？
这可以看作是一种递归！随着时间的增长，从屏幕中看到的主播人头会越来越多！
如果主播不最小化他的直播软件屏幕，那么这种递归会无休止地进行！
如果一个递归程序没有尽头，那么他就会卡掉！

认识递归

首先让大家感性地认识递归。
大家看网络直播的时候有没有见过主播的直播软件屏幕中出现了主播自己的直播软件屏幕？
这可以看作是一种递归！随着时间的增长，从屏幕中看到的主播人头会越来越多！
如果主播不最小化他的直播软件屏幕，那么这种递归会无休止地进行！
如果一个递归程序没有尽头，那么他就会卡掉！
递归的一层一层是调用系统栈，栈是一种后进先出的线性数据结构。

递归初步

递归初步

首先一定必定提到经典的 Fibonacci 数列。

递归初步

首先一定必定提到经典的 Fibonacci 数列。
设 $F_1 = 1$ $F_2 = 1$，当 $n \geq 3$ 时，有 $F_n = F_{n-1} + F_{n-2}$。

递归初步

首先一定必定提到经典的 Fibonacci 数列。
设 $F_1 = 1$ $F_2 = 1$，当 $n \geq 3$ 时，有 $F_n = F_{n-1} + F_{n-2}$。
那么如何求出任意的 F_n 呢？（不考虑时间复杂度和爆精度问题 >_<）

递归初步

首先一定必定提到经典的 Fibonacci 数列。
设 $F_1 = 1$ $F_2 = 1$，当 $n \geq 3$ 时，有 $F_n = F_{n-1} + F_{n-2}$。
那么如何求出任意的 F_n 呢？（不考虑时间复杂度和爆精度问题 >_<）
除了循环迭代以外，也可以通过一个简单的递归函数实现。

递归初步

首先一定必定提到经典的 Fibonacci 数列。
设 $F_1 = 1$ $F_2 = 1$，当 $n \geq 3$ 时，有 $F_n = F_{n-1} + F_{n-2}$。
那么如何求出任意的 F_n 呢？（不考虑时间复杂度和爆精度问题 >_<）
除了循环迭代以外，也可以通过一个简单的递归函数实现。
让我们手膜出 $x = 6$ 时，这个函数经历的变化。

递归初步

首先一定必定提到经典的 Fibonacci 数列。
设 $F_1 = 1\ F_2 = 1$，当 $n \geq 3$ 时，有 $F_n = F_{n-1} + F_{n-2}$。
那么如何求出任意的 F_n 呢？（不考虑时间复杂度和爆精度问题 >_<）
除了循环迭代以外，也可以通过一个简单的递归函数实现。
让我们手膜出 $x = 6$ 时，这个函数经历的变化。
现在对递归有一定的理解了吧！

递归的应用

递归的应用

递归的本质就是用来解决问题。

递归的应用

递归的本质就是用来解决问题。
许多递归程序实现的是上一个状态到下一个状态的转换或者大状态到小状态的转换。

递归的应用

递归的本质就是用来解决问题。
许多递归程序实现的是上一个状态到下一个状态的转换或者大状态到小状态的转换。
递归的应用及其多：以后暴力的基础 dfs，许多 $O(nlogn)$ 数据结构的构建维护，快速排序，归并排序等等。

递归的应用

递归的本质就是用来解决问题。
许多递归程序实现的是上一个状态到下一个状态的转换或者大状态到小状态的转换。
递归的应用及其多：以后暴力的基础 dfs，许多 $O(nlogn)$ 数据结构的构建维护，快速排序，归并排序等等。
递归是初学者的一大障碍，也是以后许多高级算法的坚实基础，一定要掌握!!!

作业

作业

作业就是试炼场新手村中的“过程递归”题目。大家尽力去完成。
（我也正在做诶！）

谢谢！

All Made By LaTeX