第四次上机解题报告

15081070 张雨任

一、A 题

1. 题目代号与评测记录序号：{A265120}
2. 解题思路：
   考察基本的队列操作，我用循环队列来实现。循环队列要注意开一个大小为 maxSize 的数组，只能放 maxSize-1 人，所以这里 maxSize 是 21。

二、B 题

1. 题目代号与评测记录序号：{B265766}
2. 解题思路：
   这道题的思路是按照从小到大来合并，能够使得体力最小，所以每次合并之前要把数组从小到大排序，把数组相邻两个相加作为后面的值，并且累加耗费的体力。

三、C 题

1. 题目代号与评测记录序号：{C267611}
2. 解题思路：
   这道题历经波折。
   首先，根据题意写出递推公式。对于长度为 n 的矩形，先要看大体上 n 是怎么分的，即先不考虑两个 6*3 可以组成一个 6*6 的情况。因此有二元一次方程 n=3x+6y，其中 x 代表需要的 6*3 的矩形个数，y 代表需要的 6*6 的矩形的个数。那么可以求出该方程的非负整数解，例如 n=18 的时候，根据二元一次方程可以解得，(6,0)、(4,1)、(2,2)、(0,3)为方程的解。然后从两个维度来考虑问题。称 6*3 的矩形为小矩形，6*6 的矩形为大矩形。首先，考虑摆放顺序的问题。对于 x 个大矩形和 y 个小矩形的情况，有多少种摆放方式：把 x 个大矩形放在桌子上，那么它有 x+1 个空位供 y 个小矩形来选择，那么情况数量为：C(x+y, y)种情况（C 为组合数），。其次，再考虑一个大矩形可以分成两个小矩形，对于某种拥有 y 个大矩形的情况，每个大矩形都可以分成两个小矩形，因此每个大矩形都有两种选择：分成两个小矩形和不分维持大矩形的状态，那么有 2^y 种分法。即对于 n=3x+6y 的每一个解(x,y)的情况数量为：C(x+y, y)* 2^y，再根据 x 和 y 的函数关系可整理为：C((n-3*y)/3, y)*(2^y)。而对于每一个 n，其 y 的取值可从为 0 到 n/2 的每一个整数。则给出一个 n 可以通过遍历 y 每个可能的取值，累加每个 y 对应的情况个数，递推公式可写成 sum[y]=sum[y-1]+C((n-3*y)/3, y)*(2^y)。然后信心满满地提交了，然后 WA 了。算法肯定没问题，结果也没问题。然后我检查最大可能的 n 值，也就是 n=750 的时候，炸了，超过了 unsigned long long 的范围，所以这道题是一个高精度输出问题。
   对于高精度输出，我的想法是使用结构体数组，每个结构体 situ[i]代表 n 为 i 时的情况，那么每个结构体中又有一个数组 num[]，num[i]代表第 i+1 位的数

字是多少，还有一个 int 类型的变量 digit，代表这个数有多少位。
然而高精度的运算能进行普通的四则运算就已经不错了，而我的递推公式有组合数和乘方的运算，基本上很难用上述提到的结构体来实现，所以要更改递推公式，而且这个递推公式不能有太复杂的运算。通过前面的公式，基本上 n=210 之前的数都可以算出来，所以我就算了一些数开始找规律。那么对于数对(n, f[n])（这里 f[n]是 n 对应的情况数量）有如下数对符合要求：(3,1), (6,3), (9,5), (12,11), (15,21), (18,43), (21,85), (24,171), (27, 341), (30,683)…
从中不难发现规律（的确不难……）f[n]=2*f[n-6]+f[n-3]，这就是新的递推公式，而且仅用到了乘法和加法。当然也可以这么解释这个递推公式：对于某种情况，假设为 f[i]，那么要分两种情况讨论：首先，因为 f[i-3]长度和 f[i]差了 3，所以在 f[i-3]的情况后面续一个 6*3 的小矩形，所以所有 f[i-3]的情况都可以变成 f[i]的情况，所以是 1 倍的 f[i-3]；其次对于 f[i-6]的情况，f[i-6]长度和 f[i]差了 6，就是在后面续一个大矩形（6*6），大矩形可以有三种分法：一种是直接摆大矩形，另一种是两个 6*3 横着放。不能算上两个 6*3 竖着放的，因为本身在 f[i-3]后面续小矩形中就包括了竖着放的情况了，再算就重复了，所以是 2 倍的 f[i-6]。因此递推公式是 f[n]=2*f[n-6]+f[n-3]，易得 f[3]=1，f[6]=3。
接下来就可以构造运算的函数了，首先是乘，其实这里就是加倍，那么可以从 digit=0 即第一位开始每一位都乘二，结果 mod 10 作为所求的对应位的值，如果某一位乘 2 大于 10 了，那么要进位，即前一位的值自增 1，循环往复。要注意最后一位，如果进位了的话，digit 还要自增 1，否则其实 digit 少加了一次。加法其实类似，也是注意进位就可以了。最后输出的时候，要从高位向低位输出。

3. 易错点：
   首先找递推公式是一个难点，其次要意识到 WA 是高精度的缘故而不是算法的问题。解决这两点，这道题就可以解出来了。然后写运算函数的时候要关注进位。
4. 经验：
   通过这道题可以掌握高精度的简单运算，以后遇到这种问题就可以很快地解决了。

四、D 题

1. 题目代号与测评记录序号：{D266781}
2. 解题思路：
   这道题的基本思想就是某个柱子 A 是和他后面的比它高或者和他一样高的柱子一起盛水的，而盛水量就是他们中间那些柱子和柱子 A 之间高度差之和。但是还要额外考虑一种情况，就是某柱子 B 后面没有比它还高的柱子，那么它会和他后面所有柱子中最高的柱子一起盛水。其次，不需要遍历每根柱子，比如这次算出的结果是 C 柱子和 D 柱子一起盛水，那么接下来从 D 柱子开始遍历就可以了，这样可以节省时间。最后输出累积的总和就可以了。
3. 易错点：
   容易忘记考虑“某柱子后面没有比它还高的柱子”的盛水情况，这样算出的值会缺少一些水量。

4. 经验：不要沉迷于测试数据和样例，应该从代码出发，认真读题，认真读代码，理清逻辑。

五、E 题

1. 题目代号与测评记录序号：{E268045}
2. 解题思路：
   首先这道题还是先找到递归的思路，把整个过程抽象出来。过程如下：把 n-1 个盘子从 A 经过 B 移到 C，把 A 剩下的最大的盘子移到 B，再把 n-1 个盘子从 C 经过 B 移动到 A，再把最大的盘子从 B 移动到 C，最后把 n-1 个盘子从 A 经过 B 移动到 C；n=1 的时候就是把盘子从 A 经过 B 移到 C。改成栈的话有两种实现方式，一种建立三个栈来模拟三根柱子，虽然实现了，但这种方法还是用到了递归的，所以不太符合题意。于是我用了第二种方法，在殷人昆教材上有模型，建立一个栈，这个栈就相当于递归时用到的栈，这个栈存储的是情况，把每种情况（可以用结构体，其中包含 n，A，B，C 的数据）压到栈里，然后每次弹出栈顶的情况进行操作，即用栈模拟递归。构造两种情况，tmp 和 rslt，tmp 代表临时借用，用来压进栈中，rslt 代表每次弹出的情况，对它进行操作。
   其次，来讨论算法。在汉诺塔递归的时候，产生了递归树，对于递归树的每个结点，都是先执行左分支，然后中，最后右，也就是执行分支的顺序是从左到右。而用栈，可以把结点（也就是刚刚提到的情况）按照右、中、左的顺序（从右到左）压入栈中，记录在栈中。那么在执行的时候，弹出栈顶的情况来执行，就符合递归的执行顺序（从左到右）了，这就是为什么用栈模拟递归的时候要按照执行顺序的反方向把情况压入栈中。
3. 易错点：
   题目中 n 取值范围是[0,12]，而 n 为 0 的时候无法操作，所以要单独考虑，n 为 0 的时候什么都不输出。还有栈模拟递归的时候要按照递归执行的反方向来压入栈中。然后当 rslt.m 为 1 的时候不能仅仅像递归那样直接输出 A 到 B 到 C，因为压入栈中的还有起始柱和目标柱相邻的情况，所以当起始柱和目标柱相差为 1（也就是相邻）的时候，仅仅挪动一次，否则完全按照递归去写，会多打印一些奇怪的步骤。

六、F 题

1. 题目代号与测评记录序号：{F267123}
2. 解题思路：
   这道题要找最长子序列的长度，基本思想是从左到右遍历，计算每个元素对应的符合要求的序列的长度，最后取最大值。遍历的时候要分两种情况，一种我取名为“向上波动子序列”，另一种是“向下波动子序列”，顾名思义，比如：4,5,4,4,5,4,5,5,5 属于向上波动，4,3,4,4,3,4,4,3,3 属于向下波动，因为你永远不知道从某个数开始的子序列是向上波动还是向下波动，所以我就干脆两种都遍历，取子序列长度大的作为这个数开始的真正的子序列长度。思路是这样，但是交上去会超时，原因是我遍历了根本不会产生更长的子序列的数，还是刚才的例子，4,5,4,4,5,4,5,5,5,6，第一个 4 开始的子序列长度是 9，

接下来要遍历第二个数，也就是 5，但是可以想象它的子序列长度是 8，到最后一个数 6 还是要停下来，因此 5 其实是不需要遍历的。那么哪些数需要遍历呢？继续看刚才的例子，可以看到倒数第 4 个数是需要遍历的，因为 5,5,5,6 又成为了新的子序列，而且有可能比刚才的序列长度大，那么如果子序列结束的数的下一个数（姑且称为 next）有可能和前面的数连成一个新的子序列的话，那么需要找到这个序列开始的数，并从这个数开始遍历，否则就从 next 开始遍历（比如把刚才的序列的 6 换成 7，7 就不会和前面的数连成一个子序列）。什么数有可能和前面的数连成一个新的子序列？是和上一个序列结束的数相差小于等于 1 的数。那么怎么找到这个新序列开始的数呢？继续用序列 4,5,4,4,5,4,5,5,5,6 来讲，现在 next 是 6 了，设序列结束的值是 last，那么需要回溯到之前一个不等于 last 的值。原因是要形成一个子序列要满足差小于等于 1，对于 next 来说，在前面子序列范围内是没有和 next 相等的数的，否则该子序列就不会中断了，所以符合的只会是和 next 前面的数（也就是 last）相等的值，才会在新的子序列中。这样就会节省下很多时间，只遍历有可能比之前序列长度大的序列。

3. 易错点：
   超时之后要思考怎么减少遍历的次数，分析哪些数是不可能产生更长子序列的，从而跳过这些数不进行遍历。

七、G 题

1. 题目代号与测评记录序号：{G266710}
2. 解题思路：
   （改一下练习赛的就出来了，很后悔没写这道题）
   第一步：输入一个字符串，是一个合法的表达式，首先对它进行一些处理。负号是一个很讨厌的东西，和减号混在一起，所以遍历表达式，识别负号（只有表达式开头的‘-’和右括号后面的‘-’是负号），在负号前面加个 0，就可以很舒服的运算了。然后处理**（乘方），**容易和*混淆，而且我的符号栈是 char 类型，存不了**，所以遍历表达式，把**变成^舒服一些，操作就是删除**，插入^，用 string 类的 insert 和 erase 即可。最后在表达式后面加上‘#’，便于识别表达式的开始和结束。
   第二步：建立两个栈，写好函数。建一个操作数栈，用来存放操作数，再建一个运算符栈，存放运算符。写函数：优先级函数，由于具有相同优先级的符号要服从表达式“先出现的符号先运算”的原则，所以栈内的符号优先级高于栈外的符号（对于具有相同优先级的符号来说）。运算函数，输入符号和操作数，返回结果。
   第三步：把运算符和操作数分类，同时运算。遍历表达式，遇到数字扔到数栈去，遇到符号就和栈内的操作符的优先级比较：相等说明是括号匹配或者表达式结束，弹出栈内操作符；栈内低则应该先运算栈外的运算符，所以栈外的压入栈中；栈内高则弹出栈内的运算符，和数栈弹出的两个数运算，再放回数栈中。循环往复。最后输出操作数栈里的值即可。
3. 易错点：
   这道题还是挺麻烦的，要细心。优先级栈内低于栈外，运算符之间的优先级也要保证正确。