本章为DP初步,搜索译题 - preface

Preface Numbering 序言页码

描述

一类书的序言是以罗马数字标页码的。传统罗马数字用单个字母表示特定的数值，以下是标准数字表:

```
I 1   L 50   M 1000
V 5   C 100
X 10  D 500
```

最多 3 个可以表示为 10^n 的数字(I,X,C,M)可以连续放在一起，表示它们的和:

```
III=3
CCC=300
```

可表示为 5x10n 的字符(V,L,D)从不连续出现。

除了下一个规则，一般来说，字符以递减的顺序接连出现:

```
CCLXVIII = 100+100+50+10+5+1+1+1 = 268
```

有时，一个可表示为 10^n 的数出现在一个比它大 1 级或 2 级的数前(I 在 V 或 X 前面，X 在 L 或 C 前面，等等)。在这种情况下，数值等于后面的那个数减去前面的那个数:

```
IV = 4
IX = 9
XL = 40
```

像 XD, IC, 和 XM 这样的表达是非法的，因为前面的数比后面的数小太多。对于 XD(490 的错误表达)，可以写成 CDXC; 对于 IC(99 的错误表达)，可以写成 XCIX; 对于 XM(990 的错误表达)，可以写成 CMXC。

给定 N(1 <= N < 3,500)， 序言的页码数，请统计在第 1 页到第 N 页中，有几个 I 出现，几个 V 出现，等等 (从小到大的顺序)。不要输出并没有出现过的字符。

比如 N = 5, 那么页码数为: I, II, III, IV, V. 总共有 7 个 I 出现，2 个 V 出现。

格式

PROGRAM NAME: preface

INPUT FORMAT:

(preface.in)

一个整数 N。

OUTPUT FORMAT:

(preface.out)

每行一个字符和一个数字 k，表示这个字符出现了 k 次。字符必须按数字表中的递增顺序输出。

SAMPLE INPUT

```
5
```

SAMPLE OUTPUT

```
I 7
V 2
```

题解 - preface

分析

数学问题

这道题的 n 只有 3000 多，从 1~n 把根据题目转换成字母，然后统计出现次数也不会超时。不过这里介绍另一种更高效的算法。

0~9 中 IVX 出现的次数和 10~19,20~29,x0~x9 中的相同（其它字母都未出现）。0~99 中仅十位产生的字母中，字母出现的次数相同，不同的是字母成了 XLC，后移了 2 位。

我们把 n 按十进制位从低到高的顺序统计每位出现的字母次数。

```
以 n=234 为例，
```

```
仅个位：23 个 0~9 的次数，1 个 0~3 的次数，1 个 4 的次数。(IVX)
仅十位：2 个 0~9 的次数*10，1 个 0~2 的次数*10，5 个 3 的次数。(XLC)
仅百位：0 个 0~9 的次数*100，1 个 0~1 的次数*100，35 个 2 的次数。(CDM)
```

译题 - subset

Subset Sums 集合

描述

对于从 1 到 N (1 <= N <= 39) 的连续整数集合，能划分成两个子集合，且保证每个集合的数字和是相等的。举个例子，如果 N=3，对于{1，2，3}能划分成两个子集合，他们每个的所有数字和是相等的：

```
{3} 和 {1,2}
```

这是唯一一种分法（交换集合位置被认为是同一种划分方案，因此不会增加划分方案总数） 如果 N=7，有四种方法能划分集合{1，2，3，4，5，6，7}，每一种分法的子集合各数字和是相等的:

```
{1,6,7} 和 {2,3,4,5} {注 1+6+7=2+3+4+5}
{2,5,7} 和 {1,3,4,6}
{3,4,7} 和 {1,2,5,6}
{1,2,4,7} 和 {3,5,6}
```

给出 N，你的程序应该输出划分方案总数，如果不存在这样的划分方案，则输出 0。**程序不能预存结果直接输出。**

格式

PROGRAM NAME: subset

INPUT FORMAT:

(file subset.in)

输入文件只有一行，且只有一个整数 N

OUTPUT FORMAT:

(file subset.out)

输出划分方案总数，如果不存在则输出 0。

SAMPLE INPUT

```
7
```

SAMPLE OUTPUT

```
4
```

分析

动态规划

设分成的子集为 set1, set2。

设 dp[i, j]表示前 i 个数放入 set1 中，使和为 j 的方案数。

```
dp[i, j]=dp[i-1, j]+dp[i-1, j-i]     j-i>=0
dp[i, j]=dp[i-1, j]j-i<0
```

因为 set1 和 set2 没有区别，所以对一种方案计算了 2 次，最后的结果要除 2。

从数据结构上优化：

```
for i:=1 to n do
 for j:=s(i) downto i do
    inc(dp[j], dp[j-i]);
```

{其中 s 为求和函数}

具体优化依据参见 DD 牛的《背包九讲》

分析二

递推设 ans[i, x]表示在前个 i 元素里选择若干，使其和为 x 的选法

有 ans[i, x]=ans[i-1, x]+ans[i-1, x-i]

最后的输出就是ans[n-1,((n+1)*n div 4)-n]

注意解不存在的情况单独判断

译题 - runround

Runaround Numbers 循环数

描述

循环数是那些不包括 0 这个数字的没有重复数字的整数（比如说，81362）并且同时具有一个有趣的性质，就像这个例子：

如果你从最左边的数字开始（ 在这个例子中是 8）数最左边这个数字到右边(回到最左边如果数到了最右边),你会停止在另一个新的数字(如果没有停在一个不同的数字上，这个数就不是循环数).就像：

```
8 1 3 6 2 从最左边接下去数 8 个数字：1 3 6 2 8 1 3 6 所以下一个数字是 6.
重复这样做（这次从“6”开始数 6 个数字）并且你会停止在一个新的数字上：2 8 1 3 6 2，也就是 2.
再这样做（这次数两个）：8 1;再一次（这次一个）：3;
又一次：6 2 8 这时你回到了起点，在从每一个数字开始数 1 次之后.
```

如果你在从每一个数字开始数一次以后没有回到起点，你的数字不是一个循环数。

给你一个数字 M（在 1 到 9 位之间），找出第一个比 M 大的循环数，并且一定能用一个无符号长整形数装下。

格式

PROGRAM NAME：runround

INPUT FORMAT:

(file runround.in)

仅仅一行，包括 M

OUTPUT FORMAT:

(file runround.out)

仅仅一行，包括第一个比 M 大的循环数。

SAMPLE INPUT

```
81361
```

SAMPLE OUTPUT

```
81362
```

方法 1

从开始数往后枚举，然后判断其是不是 runround number，如果是就输出退出，由于每次判断的复杂度是常数的，算法复杂度是 O(n)，不会超时。

方法 2

从开始数的位数开始，迭代搜索生成数，然后判断其是不是 runround number 且大于开始数，因为数字最多只有 9 位，且每位都不同，算法复杂度是 O(9!)，不会超时。

译题 - lamps

Party Lamps 派对灯(IOI98)

描述

在 IOI98 的节日宴会上，我们有 N(10<=N<=100)盏彩色灯，他们分别从 1 到 N 被标上号码。 这些灯都连接到四个按钮：

```
按钮 1：当按下此按钮，将改变所有的灯：本来亮着的灯就熄灭，本来是关着的灯被点亮。
按钮 2：当按下此按钮，将改变所有奇数号的灯。

按钮 3：当按下此按钮，将改变所有偶数号的灯。

按钮 4：当按下此按钮，将改变所有序号是 3*K+1(K>=0)的灯。例如：1,4,7...
```

一个计数器 C 记录按钮被按下的次数。当宴会开始，所有的灯都亮着，此时计数器 C 为 0。

你将得到计数器 C(0<=C<=10000)上的数值和经过若干操作后所有灯的状态。写一个程序去找出所有灯最后可能的与所给出信息相符的状态，并且没有重复。

格式

PROGRAM NAME: lamps

INPUT FORMAT:

(file lamps.in)

不会有灯会在输入中出现两次。

第一行: N。

第二行: C 最后显示的数值。

第三行: 最后亮着的灯,用一个空格分开，以-1 为结束。

第四行: 最后关着的灯,用一个空格分开，以-1 为结束。

OUTPUT FORMAT:

(file lamps.out)

每一行是所有灯可能的最后状态(没有重复)。每一行有 N 个字符，第 1 个字符表示 1 号灯，最后一个字符表示 N 号灯。0 表示关闭，1 表示亮着。这些行必须从小到大排列（看作是二进制数）。

如果没有可能的状态，则输出一行'IMPOSSIBLE'。

SAMPLE INPUT

```
10
1
-1
7 -1
```

在这个样例中，有 10 盏灯，只有 1 个按钮被按下。最后 7 号灯是关着的。

SAMPLE OUTPUT

```
0000000000
0101010101
0110110110
```

在这个样例中，有三种可能的状态：

所有灯都关着

1,4,7,10 号灯关着，2,3,5,6,8,9 亮着。

1,3,5,7,9 号灯关着，2， 4， 6， 8， 10 亮着。

题解 - lamps

==分析== bbbb 每个按钮按 2 次和没按效果是一样的。所以每个按钮或者按或者不按，一共有 2^4=16 中状态。枚举每个按钮是否按下，然后生成结果，排序输出即可（注意判重）。

c 的约束就是按下的按钮数<=c。

另外灯 1 和灯 7,2 和 8,3 和 9...是一样的因此当 N>=6 时只需处理前 6 个,排序时转换为 10 进制数, 输出时反复输出前 6 个的状态.

另一种判断按下按钮数是否为 C 的方法（与上面的差不多）：

在判断是否按了 c 次时，就不好直接判断了，因为我们的思路是：“每个按钮按 2 次和没按效果是一样的。所以每个按钮或者按或者不按” 但是实际上每个按钮按 2 次和没按效果有细微差别——就是按的次数。

为了解决这个问题，我从奇偶考虑：假如有两种情况结果相同，但按的次数不同，分别为 a,b。而如果 a-b 是偶数，那么按的次数少的那种情况完全可以通过按 2K 次来凑够次数（k 为正整数），因此，我们可以用 mod 语句来完成判断是否能凑够 c 次，用此方法可以发现不用判重了，因为我搜索的是基本情况，而其他的情况自然被无形的滤掉了具体过程如下：（change(k)表示按第 k 个按钮，在这里就不贴出了。）

```
function check:boolean;
var
  i:integer;
begin
  for i:=1to cy do
    if now[y[i]]<>1then exit(false);
  for i:=1to cny do
```

```
    if now[ny[i]]<>0then exit(false);
  if((c mod
2)=(tn mod
2)){检查是否能凑够 c 次}and(tn<=c)then
    exit(true)
    else exit(false);
end;

procedure  tryit(k:integer);
var
  i,j,t:integer;
begin
  if(k>4)thenbeginif check then rec; exit; end;
  change(k);
  inc(tn);
  tryit(k+1);//按
  dec(tn);
  change(k);
  tryit(k+1); //不按 end; (by ymfoi)
```