

浅谈伪随机数发生器 (pseudo-random number generator) 算法

何 唯

数学与信息学院 14 信息与计算科学 3 班 201430120308

关键词: Prng 种子 平方取中法 线性同余

一、种子

C 语言常用产生随机数函数 `rand()`，通过 `time()` 函数获取当前时间，传递给 `srand()` 函数，处理后提供给 `rand()` 函数产生随机序列。在这个例子中的当前时间 (据说是从 1970 年 1 月 1 日零时零分零秒到目前的秒数) 就是所谓的随机种子，随机种子之于伪随机数发生器，就像数列首项之于数列，是产生的随机序列的开端。当种子经过特定算法处理，就成为了一段随机序列。当然，种子不能为常量，因为我们需要每次使用的随机序列都是不一样的。例，`rand()` 函数在没有使用 `srand()` 设置种子时，默认种子是 1，然后随机序列前十个数字每次都是 41, 67, 34, 0, 69, 24, 78, 58, 62, 64。显然，这就是我们要当前时间这样的随机数作为种子的原因。每时每刻你取当前时间都是相对随机的，可以看作随机的种子。

二、取中法

定义：选择一个 m 位数 N_i 作为种子平方计算，若不足 $2m$ 个位，在前补 0。在这个数选中 m 个位的数，作为结果 N_{i+1} 。以此迭代下去。(m 只能为偶数，这样才能取中！！)

例如 $m=4$ 时的平方取中：

- 1) 将种子设为 N_0 ，并 $\text{mod } 10000$ 得到 4 位数
- 2) 将它平方得到一个 8 位数 (不足 8 位时前面补 0)
- 3) 取中间的 4 位数可得到下一个 4 位随机数 N_1
- 4) 重复 1-3 步，即可产生随机数序列

例：The seed is 1063 ($N_0=1063$)

↓

$1063 \bmod 10000 = 1063$

↓

$1063^2 = 1129969 \rightarrow 01129969$ (add 0)

↓

01/1299/69

$N_1=1299$

Then, 6874	2518	3403	5804	6864	1144	3087	5295	370	1369
8741	4050	4025	2006	240	576	3317	24	5	0
0	0	0	0	0	0	0	0		

首先，这个算法不只是有退化到 0 的情况。但是，这也是其中一种情况。由此计算可得，

结论一 若此算法序列中出现两位数或一位数 (包括种子就是两位数或一位数的情况)，必定退化到 0，且序列最多生成 8 个不为 0 的元素。

下面是证明的过程：

1、当我们输入种子(seed)为 1-9（一位数）时，易得退化为零。

例： $9*9=81 \rightarrow 00/0000/81 \rightarrow 0$

此时，序列中并没有生成任何随机数。

2、当 $seed \in [10, 99]$ 时，可分成八层嵌套来讨论。

(1) 显然，当 $seed \in [10, 31]$ ， $seed^2 = 00000ix_1x_2$ ($i, x_1, x_2 = 0, 1, 2 \dots 9$)

例：

$10*10=100 \rightarrow 00/0001/00 \rightarrow 1$ $seed \in [10, 14]$ 时， $i = 1$

$15*15=225 \rightarrow 00/0002/25 \rightarrow 2$ $seed \in [15, 17]$ 时， $i = 2$

以此类推，

$31*31=961 \rightarrow 00/0009/61 \rightarrow 9$ $i = 9$

下面就回到了第一层的计算，变为一位数来考虑

$i^2 \leq 81 < 100$ ，显然也会退化为零

这里称为第一层

(2) $seed \in [32, 56]$ ， $seed^2 = 0000ix_1x_2$ ($i \in [10, 31], x_1, x_2 = 0, 1, 2 \dots 9$)

例： $32*32=00/0010/24 \rightarrow$ （下面又回到第二层的计算） $10*10=00/0001/00 \rightarrow 1 \rightarrow 0$

即， $seed \in [32, 38]$ ， $i=10$

.....

$56*56=00/0031/36 \rightarrow 31*31=961 \rightarrow 00/0009/61 \rightarrow 9 \rightarrow 0$

这里称为第二层

(3) 由此可通过计算得， $seed \in [57, 75]$ 为第三层； $seed \in [76, 87]$ 为第四层； $seed \in [88, 93]$ 为第五层； $seed \in [94, 96]$ 为第六层； $seed \in [97, 98]$ 为第七层； $seed=99$ 为第八层。

结论二：除退化为零外，随机数序列还出现随机数序列循环和收敛的情况。

还是以 $m=4$ 为例，三种收敛的情况，分别为：

1、 $100 \rightarrow 100 \rightarrow 100 \rightarrow 100 \rightarrow 100$

2、 $2500 \rightarrow 2500 \rightarrow 2500 \rightarrow 2500 \rightarrow 2500$

3、 $7600 \rightarrow 7600 \rightarrow 7600 \rightarrow 7600 \rightarrow 7600$

首先，讨论收敛的情况，通过用 C 语言编程计算可以得到属于这三种收敛的种子个数以及数值。（具体数据见附录）

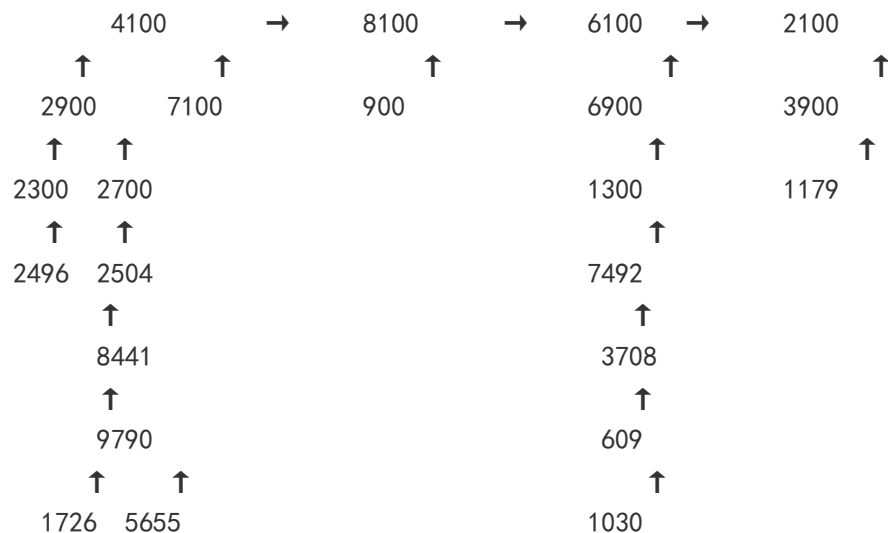
还会出现三种循环的情况，分别为：

1、 $2100 \rightarrow 4100 \rightarrow 8100 \rightarrow 6100 \rightarrow 2100$

2、 $1600 \rightarrow 5600 \rightarrow 3600 \rightarrow 9600 \rightarrow 1600$

3、 $0540 \rightarrow 2916 \rightarrow 5030 \rightarrow 3009 \rightarrow 0540$

然后，由于循环出现的几率比较大，计算机计算比较麻烦，所以观察大量数据寻找这三种循环的出现规律，有初步发现：



观察 2, 3 循环亦有类似规律。

猜想 1: 对于序列中的入口数 (4100, 8100, 6100, 2100), 总存在特定序列 (路径), 种子经算法沿路径到达入口数, 最终进入循环。

猜想 2: 序列由入口数作为树根, 分别是 4 个 r 叉树 (分支点有若干个)

推广: $m=4$ 的情况可以推广到 $m=6$ 位, $m=8$ 位.

例如

$m=6$ 时, 三位数以内必退化为零, 各层嵌套区间

[1, 31] [32, 178] [179, 423] [424, 651] [652, 807].....

$m=8$ 时, 四位数以内必退化为零, 各层嵌套区间为 [1, 99] [100, 999] [1000, 3162].....

推广一: 最多可以生成 2^m 个不为零的元素, $m/2$ 位数 (及以内) 必退化为零。

推广二: 除退化为零外, 随机数序列还出现随机数序列循环和收敛的情况。

二、LCG (linear congruential generator) 线性同余法

古老的 LCG (linear congruential generator) 代表了最好最朴素的伪随机数产生器算法。主要原因是容易理解, 容易实现, 而且速度快。LCG 算法数学上基于公式:

$$X(n+1) = (a * X(n) + c) \% m$$

其中, 各系数为:

模 m , $m > 0$

系数 a , $0 < a < m$

增量 c , $0 \leq c < m$

原始值 (种子) $0 \leq X(0) < m$

其中参数 c , m , a 比较敏感, 或者说直接影响了伪随机数产生的质量。一般而言, 高 LCG 的 m 是 2 的指数次幂 (一般 2^{32} 或者 2^{64}), 因为这样取模操作截断最右的 32 或 64 位就可以了。多数编译器的库中使用了该理论实现其伪随机数发生器 `rand()`。C 语言的 `rand()` 函数

使用了 `srand()` 提供的种子以及线性同余算法，具体通过代码验证了，见附录。Java `Random` 类被封装 无法看 `nextInt()` `nextFloat()` 源代码, 但是估计也是使用了线性同余。

下面是部分编译器使用的各个参数值：

Source	m	a	c	rand() / Random(L)的种子 位
Numerical Recipes	2 ³²	1664525	1013904223	
Borland C/C++	2 ³²	22695477	1	位30..16 in rand(), 30..0 in lrand()
glibc (used by GCC)	2 ³²	1103515245	12345	位30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++	2 ³²	1103515245	12345	位30..16
Borland Delphi, Virtual Pascal	2 ³²	134775813	1	位63..32 of (seed * L)
Microsoft Visual/Quick C/C++	2 ³²	214013	2531011	位30..16
Apple CarbonLib	2 ³¹ -1	16807	0	见Park-Miller 随机数发生器

四、梅森旋转算法（Mersenne twister）

梅森旋转算法是 Makoto Matsumoto（松本）和 Takuji Nishimura（西村）于 1997 年开发的伪随机数产生器，基于有限二进制字段上的矩阵线性再生。可以快速产生高质量的伪随机数，修正了古老随机数产生算法的很多缺陷。常见的有两个变种 Mersenne Twister MT19937 和 Mersenne Twister MT19937-64。

说到梅森旋转算法，不得不提一下线性反馈移位寄存器(LFSR)，线性反馈移位寄存器(linear feedback shift register, LFSR) 是指，给定前一状态的输出，将该输出的线性函数再用作输入的移位寄存器。异或运算是最常见的单比特线性函数：对寄存器的某些位进行异或操作后作为输入，再对寄存器中的各比特进行整体移位。

首先，移位寄存器包括两个部分

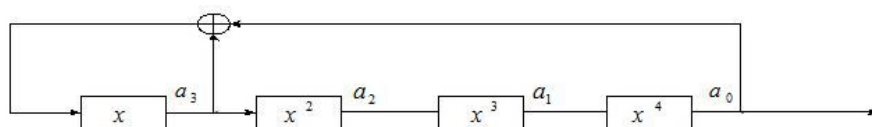
(1) 级，每一级包含一个比特，比如 11010110 是一个 8 级的移位寄存器产生的

(2) 反馈函数，线性反馈移位寄存器的反馈函数是线性的，非线性反馈移位寄存器的反馈函数是非线性的

一个 n 级的移位寄存器产生的序列的最大周期为 $2^n - 1$ ，当然这个最大周期跟反馈函数有很大关系，线性反馈函数实际上就是这个 n 级的移位寄存器选取“某些位”进行异或（二元运算，相同则返回 0，不同则返回 1）后得到的结果，这里的“某些位”的选取很重要，得到线性反馈函数之后，把这个移位寄存器的每次向右移动一位，把最右端的作为输出，把“某些位”的异或结果作为输入放到最左端的那位，这样所有的输出对应一个序列，这个序列叫做 M 序列，是最长线性移位寄存器序列的简称。

上面“某些位”的选取问题还没有解决，那么应该选取哪些位来进行异或才能保证最长周期为 $2^n - 1$ ，这是一个很重要的问题。选取的“某些位”构成的序列叫做抽头序列，理论表明，要使 LFSR 得到最长的周期，这个抽头序列构成的多项式加 1 必须是一个本原多项式，也就是说这个多项式不可约，比如 $f(x) = x^4 + x + 1$ 。

下面以一个 4 位的线性反馈移位寄存器为例说明它的工作原理。



a_3	a_2	a_1	a_0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
1	0	0	1
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

如图，假设有种子 1000，经过 4 位的线性反馈移位寄存器用梅森旋转算法计算，按

$f(x) = x^4 + x + 1$ 这个四阶本原多项式， $a_0 \rightarrow x^4$, $a_1 \rightarrow x^3$, $a_2 \rightarrow x^2$, $a_3 \rightarrow x$ ，由于

本原多项式只有 x^4 , x 系数不为 0，于是 a_0 和 a_3 进行异或运算，产生结果赋给 a_3 ； a_3 的

值给 a2，依次移位。所以 1000 得到了 1100，1100 得到了 1110. 可以看出周长为 15。在这一个周期里面涵盖了开区间 $[1, 2^4 - 1]$ 内的所有整数，并且都是没有固定顺序出现的，有很好的随机性。

表 10-1 常用本原多项式

n	本原多项式		n	本原多项式	
	代 数 式	8 进数表示法		代 数 式	8 进数表示法
2	$x^2 + x + 1$	7	14	$x^{14} + x^{10} + x^6 + x + 1$	42103
3	$x^3 + x + 1$	13	15	$x^{15} + x + 1$	100003
4	$x^4 + x + 1$	23	16	$x^{16} + x^{12} + x^3 + x + 1$	210013
5	$x^5 + x^2 + 1$	45	17	$x^{17} + x^3 + 1$	400011
6	$x^6 + x + 1$	103	18	$x^{18} + x^7 + 1$	1000201
7	$x^7 + x^3 + 1$	211	19	$x^{19} + x^5 + x^2 + x + 1$	2000047
8	$x^8 + x^4 + x^3 + x^2 + 1$	435	20	$x^{20} + x^2 + 1$	4000011
9	$x^9 + x^4 + 1$	1021	21	$x^{21} + x^2 + 1$	10000005
10	$x^{10} + x^3 + 1$	2011	22	$x^{22} + x + 1$	20000003
11	$x^{11} + x^2 + 1$	4005	23	$x^{23} + x^5 + 1$	40000041
12	$x^{12} + x^6 + x^4 + x + 1$	10123	24	$x^{24} + x^7 + x^2 + x + 1$	100000207
13	$x^{13} + x^4 + x^3 + x + 1$	20033	25	$x^{25} + x^3 + 1$	200000011

Mersenne Twister 有很多长处，例如：周期 $2^{19937} - 1$ 对于一般的应用来说，足够大了，序列关联比较小，能通过很多随机性测试。

Python 中的随机数模块 random 就是采用了梅森旋转算法来产生伪随机数列，C++11 中也有梅森旋转算法实现的随机数生成器。

五、总结

对于某个特殊的计算机随机数生成算法，只要是伪随机数生成的算法，当我们知道种子以及对应的算法，必定可以模拟整个随机序列生成过程，得到与实际生成的随机序列一致。最重要的两点就是种子和伪随机数生成器的算法。种子的话，即使是用系统时间作为种子，我们也可以设定定时器，重新调节系统时间，到达目标时间时获取种子，然后执行算法。算法的话，可以先尝试用常用的几种算法模拟过程。可能的话，可能可以通过反编译，窃取源代码文件等手段获取种子和算法。所以说伪随机数的生成方式涉及信息安全，密码学等领域，是十分值得重视的，一个好的伪随机数算法也必须得到封装保护。在成本允许的情况下，使用真随机数也是可以的。

附录：

1-5000 (收敛到 100)

```
C:\Users\v5-573\Desktop\PIA\搜索种子.exe
收敛到100的序列有48个
是种子为100
是种子为354
是种子为437
是种子为636
是种子为700
是种子为755
是种子为837
是种子为995
是种子为1005
是种子为1207
是种子为1253
是种子为1330
是种子为1394
是种子为1578
是种子为1909
是种子为2033
是种子为2211
是种子为2336
是种子为2635
是种子为2654
是种子为3141
是种子为3199
是种子为3328
是种子为3522
是种子为3539
是种子为3743
是种子为3751
是种子为3834
是种子为3937
是种子为4044
是种子为4094
是种子为4293
是种子为4298
是种子为4300
是种子为4568
是种子为4608
是种子为4728
是种子为4742
是种子为4746
是种子为4771
是种子为4865
是种子为4900
是种子为4926
是种子为4939
是种子为4947
是种子为4949
是种子为4993
是种子为4999
是种子为5000
```

```
C:\Users\v5-573\Desktop\PIA\搜索种子.exe
是种子为4900
是种子为4926
是种子为4939
是种子为4947
是种子为4949
是种子为4993
是种子为4999
是种子为5000
```

5000-9999 (收敛到 100)

```
C:\Users\v5-573\Desktop\PIA\搜索种子.exe
收敛到100的序列有56个
是种子为5001
是种子为5007
是种子为5100
是种子为5141
是种子为5245
是种子为5288
是种子为5336
是种子为5437
是种子为5586
是种子为5609
是种子为5647
是种子为5700
是种子为5737
是种子为5808
是种子为5906
是种子为5922
是种子为5977
是种子为6343
是种子为6433
是种子为6442
是种子为6491
是种子为6598
是种子为6682
是种子为6995
是种子为7005
是种子为7034
是种子为7176
是种子为7218
是种子为7245
是种子为7328
是种子为7382
是种子为7541
是种子为7608
是种子为7624
是种子为7689
是种子为7784
是种子为8271
是种子为8422
是种子为8440
是种子为8637
是种子为8658
微软拼音简捷半：
```

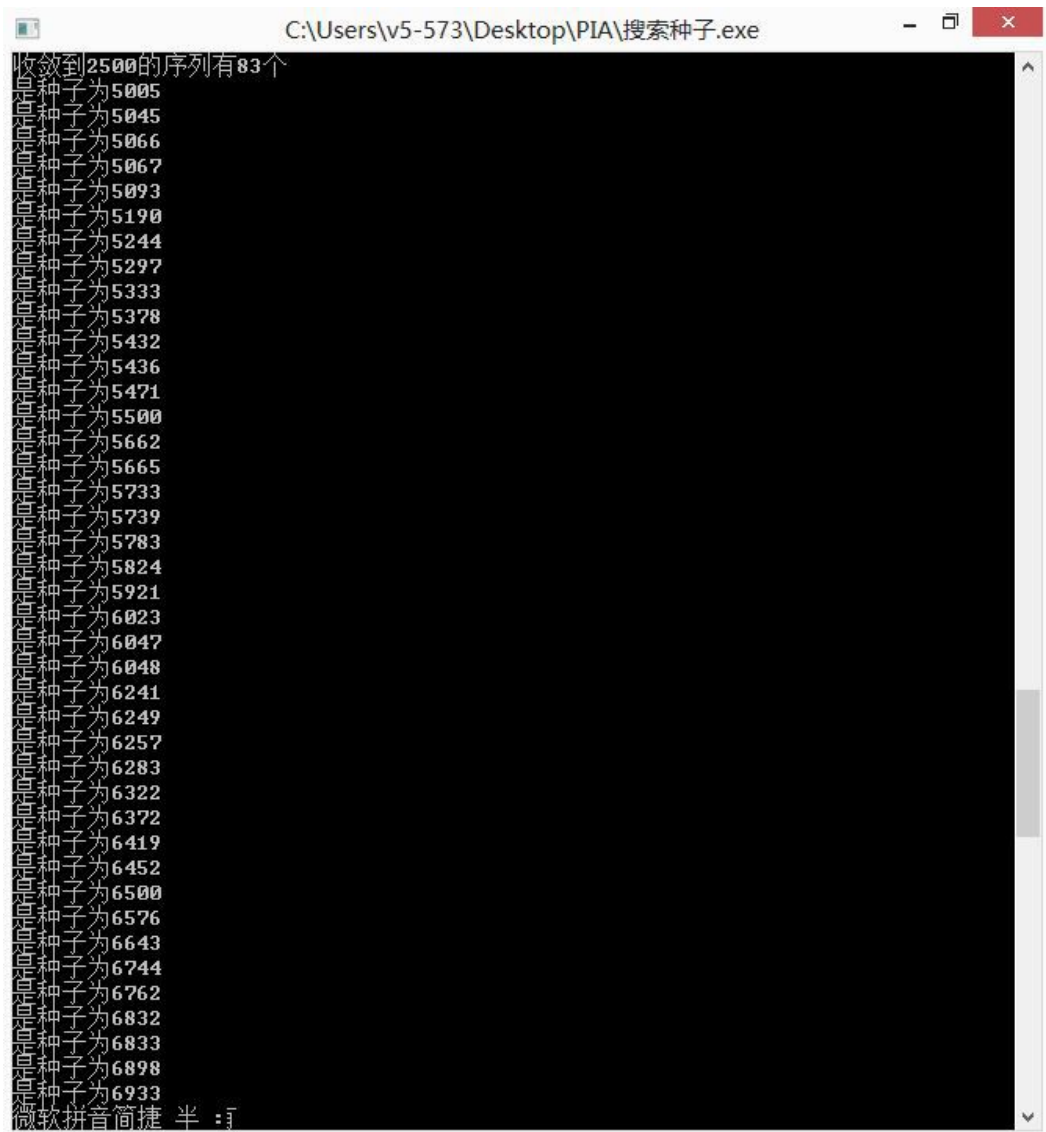
```
C:\Users\v5-573\Desktop\PIA\搜索种子.exe
是种子为8666
是种子为8747
是种子为8808
是种子为8816
是种子为8885
是种子为8886
是种子为9131
是种子为9300
是种子为9406
是种子为9432
是种子为9609
是种子为9626
是种子为9629
是种子为9832
是种子为9900
```


1-5000 (收敛到 2500)

```
C:\Users\v5-573\Desktop\PIA\搜索种子.exe
收敛到2500的序列有47个
是种子为500
是种子为582
是种子为664
是种子为790
是种子为922
是种子为923
是种子为969
是种子为1196
是种子为1205
是种子为1245
是种子为1294
是种子为1412
是种子为1500
是种子为1533
是种子为1559
是种子为1581
是种子为1607
是种子为1610
是种子为1842
是种子为2035
是种子为2437
是种子为2500
是种子为2719
是种子为2765
是种子为2910
是种子为2955
是种子为3287
是种子为3387
是种子为3500
是种子为3797
是种子为3929
是种子为4055
是种子为4172
是种子为4304
是种子为4370
是种子为4377
是种子为4408
是种子为4430
是种子为4500
是种子为4520
是种子为4535
微软拼音简捷 半 :
```

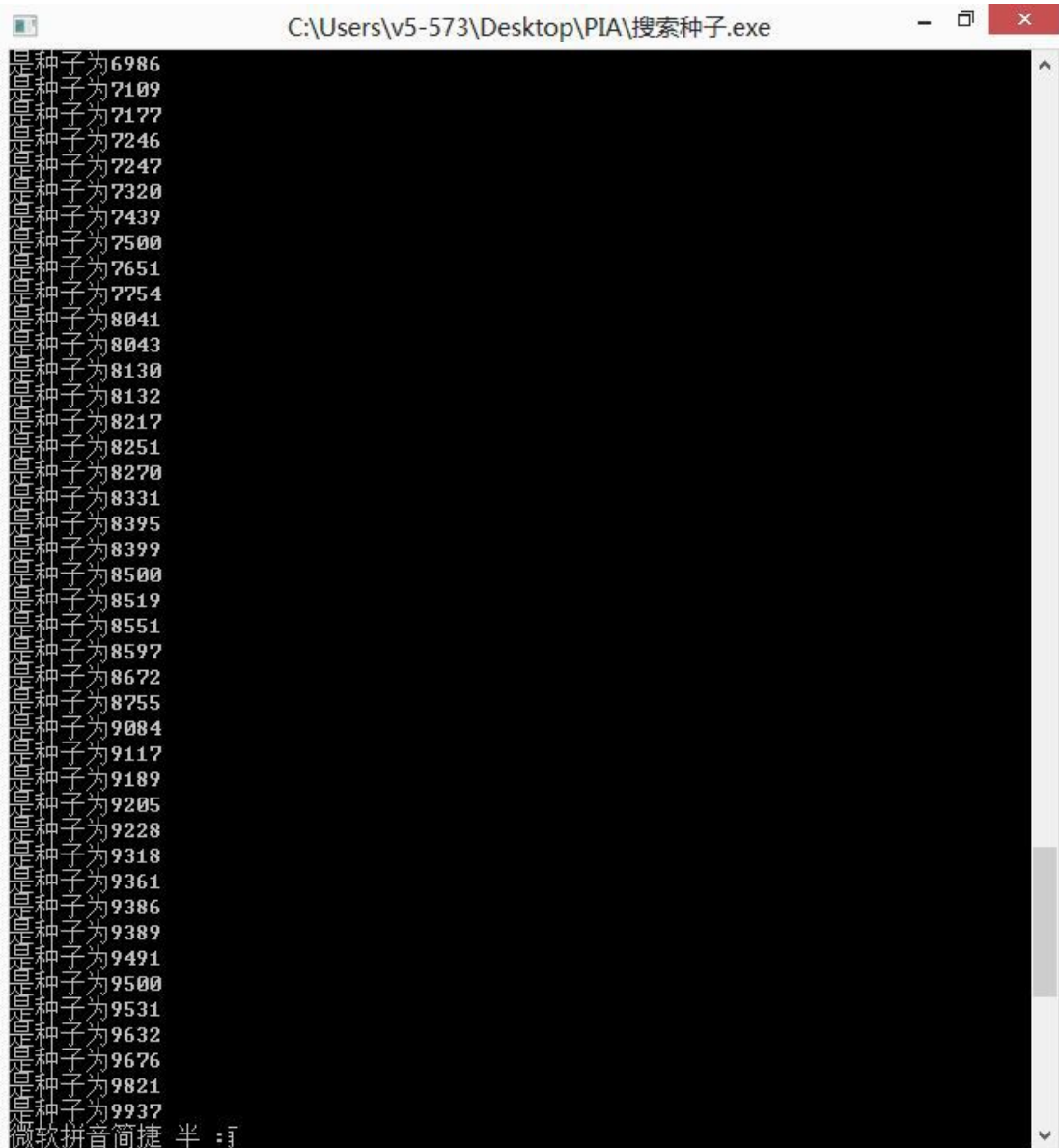
```
是种子为4681
是种子为4717
是种子为4760
是种子为4792
是种子为4815
是种子为4995
```

5000-9999 (收敛到 100)



A screenshot of a Windows command prompt window titled "C:\Users\v5-573\Desktop\PIA\搜索种子.exe". The window displays a list of 83 seed values, each preceded by the text "收敛到2500的序列有83个" and "是种子为". The seed values range from 5005 to 6933. At the bottom of the window, the text "微软拼音简捷 半" is visible.

```
收敛到2500的序列有83个
是种子为5005
是种子为5045
是种子为5066
是种子为5067
是种子为5093
是种子为5190
是种子为5244
是种子为5297
是种子为5333
是种子为5378
是种子为5432
是种子为5436
是种子为5471
是种子为5500
是种子为5662
是种子为5665
是种子为5733
是种子为5739
是种子为5783
是种子为5824
是种子为5921
是种子为6023
是种子为6047
是种子为6048
是种子为6241
是种子为6249
是种子为6257
是种子为6283
是种子为6322
是种子为6372
是种子为6419
是种子为6452
是种子为6500
是种子为6576
是种子为6643
是种子为6744
是种子为6762
是种子为6832
是种子为6833
是种子为6898
是种子为6933
微软拼音简捷 半
```



1-5000 (收敛到 7600)

收敛到7600的序列有29个

是种子为675
是种子为966
是种子为1272
是种子为1357
是种子为1618
是种子为1800
是种子为1921
是种子为2064
是种子为2087
是种子为2155
是种子为2400
是种子为2498
是种子为2502
是种子为2600
是种子为3075
是种子为3200
是种子为3218
是种子为3447
是种子为3555
是种子为3715
是种子为3936
是种子为4123
是种子为4166
是种子为4178
是种子为4296
是种子为4518
是种子为4556
是种子为4736
是种子为4920

5000-9999 (收敛到 7600)

收敛到7600的序列有31个

是种子为5064
是种子为5271
是种子为5603
是种子为6179
是种子为6265
是种子为6376
是种子为6380
是种子为6440
是种子为6458
是种子为6533
是种子为6800
是种子为6902
是种子为7044
是种子为7057
是种子为7400
是种子为7571
是种子为7600
是种子为7834
是种子为7861
是种子为7953
是种子为8012
是种子为8026
是种子为8200
是种子为8328
是种子为8405
是种子为8414
是种子为8818
是种子为9331
是种子为9560
是种子为9706
是种子为9991

Process returned 0 (0x0) execution time : 45.938 s
Press any key to continue.

平方取中.c

```
#include<stdio.h>

int main()
{
    unsigned int seed=0,temp;
    int i,j,first,number1[5000][100] = {0};
    FILE *fp;
    fp=fopen("d:\\a.txt","w");
    for(j=0;j<5000;j++)
    {
        if(j==0){printf("Enter the seed:");
        scanf("%d",&seed);
        first=seed;}
        else seed++;
        printf("The seed is :%d\n",seed);
        for(i=0;i<100;i++,temp=0)
        {
```

```

        if(i==0)
        {temp=(seed%10000)*(seed%10000);
        number1[j][0]=(temp/100)%10000;}
        else
        {temp=(number1[j][i-1]%10000)*(number1[j][i-1]%10000);
        number1[j][i] = (temp/100)%10000;}
    }printf("\n");
    }
    for(j=0;j<5000;j++)
    {
        printf("The seed is :%d\n",first);
        fprintf(fp,"The seed is %d\n",first);
        for(i=0;i<100;i++)
        {
            printf("%d\t",number1[j][i]);
            fprintf(fp,"%d\t",number1[j][i]);
        }fprintf(fp,"\n\n");
        first++;
    }

fclose(fp);
return 0;
}

```

搜索种子.c

```

#include<stdio.h>
int main()
{
    unsigned int seed=0,temp;
    int i,j,first,count1=0,count2=0,count3=0,number[5000][100] =
    {0},number1[100]={0},number2[100]={0},number3[100]={0};
    //FILE *fp;
    //fp=fopen("d:\\a.txt","w");
    for(j=0;j<5000;j++)
    {
        if(j==0){printf("Enter the seed:");
        scanf("%d",&seed);
        first=seed;}
        else seed++;
        printf("The seed is :%d\n",seed);
        for(i=0;i<100;i++,temp=0)
    {
        if(i==0)
        {temp=(seed%10000)*(seed%10000);
        number[j][0]=(temp/100)%10000;}

```

```

else
    {temp=(number[j][i-1]%10000)*(number[j][i-1]%10000);
      number[j][i] = (temp/100)%10000;}
}printf("\n");
}
for (j=0; j<5000; j++)
{
    printf("The seed is :%d\n", first);
    //fprintf(fp, "The seed is %d\n", first);
    for (i=0; i<100; i++)
    {
        printf("%d\t", number[j][i]);
        if (number[j][i]==100)
        {
            number1[count1]=first;
            count1++;
            break;}
        if (number[j][i]==2500)
        {number2[count2]=first;
          count2++;
          break;}
        if (number[j][i]==7600)
        {
            number3[count3]=first;
            count3++;
            break;}
        //fprintf(fp, "%d\t", number[j][i]);
    }//fprintf(fp, "\n\n");
    first++;
}

printf("收敛到 100 的序列有%d 个\n", count1);
for (i=0; i<count1; i++)
    printf("是种子为%d\n", number1[i]);
printf("收敛到 2500 的序列有%d 个\n", count2);
for (i=0; i<count2; i++)
    printf("是种子为%d\n", number2[i]);
printf("收敛到 7600 的序列有%d 个\n", count3);
for (i=0; i<count3; i++)
    printf("是种子为%d\n", number3[i]);
//fclose(fp);
return 0;
}

```

验证 C 语言 rand() 使用线性同余算法

```
#include<stdlib.h>
#include<stdio.h>
#define a 22695477
#define c 1
#define m 2^32
int main()
{
    int number[99999] = {0};
    int number1[99999] = {0};
    int i;
    unsigned int seed=0;
    //scanf("%d",&seed);/*手动输入种子*/
    seed=time(NULL);
    srand(seed);
    number1[0]=(a*seed+c)%m;
    printf("当前种子为: %d\n", seed);
    printf("rand() 生成的 10 个随机数: \n");
    for(i = 0; i < 10; i++)
    {

        number[i] = rand() % 100; /*产生 100 以内的随机整数*/
        printf("%d\n", number[i]);

    }
    printf("\n");
    printf("根据线性同余算法生成的 10 个随机数: \n");
    for(i=0; i<10; i++)
    {
        if(i==0)
            number1[0]=(a*seed+c)%m;
        else number1[i] = (a*number1[i-1]+c)%m;
        printf("%d\n", number1[i]);

    }
    printf("\n");

    return 0;
}
```

参考: <http://blog.csdn.net/ACdreamers/article/details/44656743>