算法设计与分析

实验报告书

实验名称：实验一　求最大公约数

学　　号：唐相儒

姓　　名：2016210894

实验时间：2018 年 3 月 15 日

目录

一 实验目的和要求

实验目的：

(1)复习数据结构课程的相关知识，实现课程间的平滑过渡；

(2)掌握并应用算法的数学分析和后验分析方法；

(3)理解这样一个观点：不同的算法能够解决相同的问题，这些算法的解题思路不同，复杂程度不同，解题效率也不同。

实验要求：

（1）至少设计出三个版本的求最大公约数算法；

（2）上机实现算法，随机产生 1000 个样本，并用计数法和计时法分别测算算法的平均运行时间 T（k）或平均计数

（3）绘制各个算法执行时间的趋势图：给定样本取值范围（规模 k），画出 T（k）的趋势图，k=100,1000,5000,10000,50000,...。

（4）对比三种 T（k）曲线，分析对比得出结论，撰写实验报告。

（5）扩展：推广的欧几里得公式

二 实验内容

1. 实现三个版本的求最大公约数算法
2. 随机产生两个 1000 维数组，分别求其公约数，按纳秒级别计时，测算运行时间
3. 绘制曲线对比实验效果
4. 实现拓展，用推广欧几里得公式求方程的解

三 实验环境

MAC OSX 10.11.6

Apple LLVM version 8.0.0 (clang-800.0.38)

Target: x86_64-apple-darwin15.6.0

Thread model: posix

VScode

编译命令：g++ -std=c++11 -o main main.cc

四 设计思想及实验步骤

1.如何在 1000 以内取随机数：

C++的随机数函数 rand 是一个伪随机数，从固定的初始种子计算得出。所以每次运行获取到的随机数序列是相同的。要解决这一问题，需要设置一个随机数种子，一般用当前时间作为种子。要控制随机数的范围，可以使用模除操作，即%运算。要控制结果在 0~1000 之间，可以使用代码：rand()%1001

```
#include<iostream>
#include <cstdlib>
#include <ctime>
int main()
{
    srand(time(NULL));
    int a = rand()%1001;
    cout <<a;
}
```

图 1：取随机数程序

2. 求最大公约数方法

最大公因数，也称最大公约数、最大公因子，指两个或多个整数共有约数中最大的一个。

方法一：蛮力法--穷举法（也叫枚举法），穷举法求两个正整数的最大公约数的解题步骤：从两个数中较小数开始由大到小列举，直到找到公约数立即中断列举，得到的公约数便是最大公约数 。而对两个正整数 a,b 如果能在区间[a,0]或[b,0]内能找到一个整数 temp 能同时被 a 和 b 所整除，则 temp 即为最大公约数。

算法 1：蛮力法求最大公约数
输入两个正整数 m，n;
输出：最大公约数 i
① 令常量 factor = 1;循环变量 i 从 2~min(m,n);
② 如果 i 是 m 和 n 的公因子,则执行③;
③ factor = factor*i; m = m/i; n = n/i;
④ 如果 i 不是 m 和 n 的公因子，则 i = i +1;

方法二：辗转相除法，也叫欧几里德算法。

设两数为 a、b(a>b)，求 a 和 b 最大公约数(a，b)的步骤如下：用 a 除以 b，得 a÷b=q……r1(0≤r1)。若 r1=0，则(a，b)=b；若 r1≠0，则再用 b 除以 r1，得 b÷r1=q……r2 (0≤r2).若 r2=0，则(a，b)=r1，若 r2≠0，则继续用 r1 除以 r2，……如此下去，直到能整除为止。其最后一个除数即为(a，b)。

算法 2：辗转相除法求最大公约数
输入两个正整数 m，n;
输出：最大公约数 i
① 求出两个数的最大值 Max 和最小值 Min;
② 计算 Max 除以 Min 所得的余数 r;
③ Max=Min,Min=r;
④ 若 r＝0,则 m,n 的最大公约数等于 Max；否则转到①;

方法三：更相减损法

更相减损术是出自《九章算术》的一种求最大公约数的算法，它原本是为约分而设计的，但它适用于任何需要求最大公约数的场合。

《九章算术》是中国古代的数学专著，其中的“更相减损术”可以用来求两个数的最大公约数，原文是：“可半者半之，不可半者，副置分母、子之数，以少减多，更相减损，求其等也。以等数约之。”就是两数先整数 2，不能整数了，就大数减去小数，用差和小数相比，再大数减去小数，知道差和小数相等。

有两整数 m 和 n：

算法 3：更相减损法求最大公约数
输入两个正整数 m，n;
输出：最大公约数 i
① 若 m>n，则 m=m-n;
② 若 m<n，则 n=n-m;
③ 若 m=n，则 m（或 n）即为两数的最大公约数;
④ 若 a≠b，则再回去执行①.

3.计时问题

在 Intel Pentium 以上级别的 CPU 中，有一个称为“时间戳（Time Stamp）”的部件，它以 64 位无符号整型数的格式，记录了自 CPU 上电以来所经过的时钟周期数。由于目前的 CPU 主频都非常高，因此这个部件可以达到纳秒级的计时精度。这个精确性是上述两种方法所无法比拟的。在 Pentium 以上的 CPU 中，提供了一条机器指令 RDTSC（Read Time Stamp Counter）来读取这个时间戳的数字，并将其保存在 EDX:EAX 寄存器对中。由于 EDX:EAX 寄存器对恰好是 Win32 平台下 C++语言保存函数返回值的寄存器，所以我们可以把这条指令看成是一个普通的函数调用。

但是由于我的电脑是 mac，所以使用一种 linux 中用户空间获得 ns 级时间的方法：使用 clock_gettime 函数，函数原型如下：

```
long sys_clock_gettime (clockid_t which_clock, struct timespec *tp);
```

which_clock 参数解释：

CLOCK_REALTIME:系统实时时间,随系统实时时间改变而改变,即从 UTC1970-1-1 0:0:0 开始计时,中间时刻如果系统时间被用户该成其他,则对应的时间相应改变

CLOCK_MONOTONIC:从系统启动这一刻起开始计时,不受系统时间被用户改变的影响

CLOCK_PROCESS_CPUTIME_ID:本进程到当前代码系统 CPU 花费的时间

CLOCK_THREAD_CPUTIME_ID:本线程到当前代码系统 CPU 花费的时间

struct timespec 结构:

```
struct timespec
{
 time_t tv_sec;
 long int tv_nsec;
};
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
struct timespec
{
 time_t tv_sec;
 long int tv_nsec;
};
int main(void)
{
   struct timespec time_start={0, 0},time_end={0, 0};
   clock_gettime(CLOCK_REALTIME, &time_start);
   printf("start time %llus,%llu ns\n", time_start.tv_sec, time_start.tv_nsec);
   clock_gettime(CLOCK_REALTIME, &time_end);
   printf("end   time %llus,%llu ns\n", time_end.tv_sec, time_end.tv_nsec);
   printf("duration:%llus %lluns\n", time_end.tv_sec-time_start.tv_sec, time_end.tv_nsec-time_start.tv_nsec);
   return 0;
}
```

图 2：clock_gettime 计时程序

4.拓展欧几里得算法

问题：ax+by=c，已知 a、b、c，求解使该等式成立的一组 x，y。其中 a、b、c、x、y 均为整数.

a,b 的最大公约数为 gcd(a,b)。如果 c 不是 gcd(a,b)的倍数，则该等式无解，因为等式左边除以 gcd(a,b)是整数，而等式右边除以 gcd(a,b)后为小数。

因此，只有当 c 是 gcd(a,b)的倍数的时候，该等式有解。这样，可以通过计算使 ax1+by1=gcd(a,b)成立的 x1、y1，然后有 x=(c/gcd(a,b))*x1，y=(c/gcd(a,b))*y1，得到 x,y。

问题就被转换为求使 ax+by=gcd(a,b)成立的一组 x,y。这可以用扩展欧几里德算法求解。如下：

如果 b 为零，则 gcd(a,b)=a，那么 x=1,y=0 为一组解。

如果 b 不为零，根据欧几里德定理，可以设

ax1+by1=gcd(a,b)=gcd(b,a%b)=bx2+(a%b)y2=bx2+(a-(a/b)*b)y2

化简后有 x1=y2，y1=x2-(a/b)y2。因此 x1,y1 依赖于 x2,y2，同理依次类推递归调用求出 x3,y3,x4,y4……，类似于辗转相除，直到 b=0 时，求出 xn,yn，便可以推出 x1,y1 的值。

```
int exgcd(int a,int b)
{                                   //扩展欧几里得算法
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    int r=exgcd(b,a%b);
    int t=x;
    x=y;
    y=t-a/b*y;
    return r;
}
```

图 3：拓展欧几里得程序示例

五 实验结果及分析

在实验中，我取了 100、500、1000、3000、5000、7000、10000、30000、50000 这 9 组数据，分别测算时间，结果如下：

	1	2	3	4
1		蛮力法	辗转相除	更相减损法
2	100	0.000193	0.000001	0.000051
3	500	0.000732	0.000003	0.000092
4	1000	0.001436	0.000084	0.000137
5	3000	0.004554	0.000095	0.000172
6	5000	0.007046	0.000101	0.000188
7	7000	0.009947	0.000105	0.000206
8	10000	0.012961	0.000109	0.000217
9	30000	0.039228	0.000122	0.000244
10	50000	0.068877	0.000129	0.000265

图 4：三种算法所用时间按照随机数大小增长而增长的表

可视化效果如下：

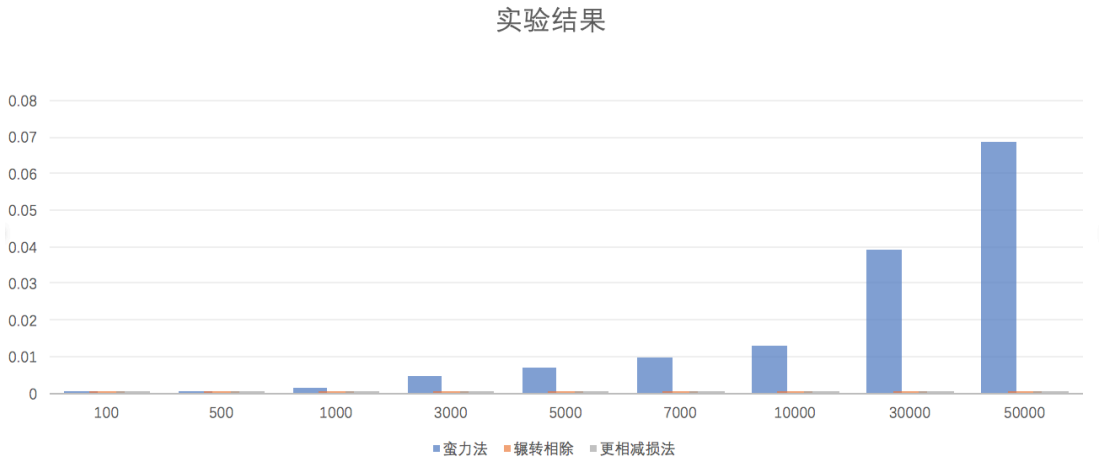

图 5：柱状图对比时间

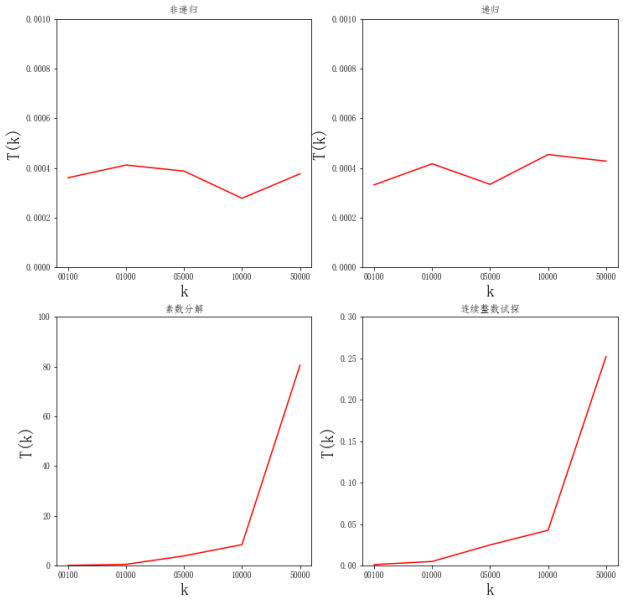

图 6：单独算法使用连续整数试探的耗时示意图

分析：

1. 从平均时间和平均次数两张图标上可以看出，各个算法所用时间和运算次数的增长是一致的。
2. 欧几里得算法是三个算法中最高效的，随着规模的不断增大，欧几里得算法的斜率几乎为 0，它的时间花费和运算次数没有明显的增加，基本上保持不变，可见欧几里得算法的高效性。
3. 蛮力法效率是最低的，并且随着规模的不断增大，其耗时和运算次数增长很快。
4. 可见，在时间复杂度上面，蛮力法远大于更相减损法和辗转相除法，而辗转相除法时间复杂度略大于辗转相除法。

六 实验体会（包括本实验中遇到的问题、具体的解决方法、还没有解决的问题、实验收获等）

1.辗转相除法与更相减损术求最大公约数的联系与区别：(1)都是求最大公约数的方法.计算上辗转相除法以除法为主.更相减损术以减法为主.计算次数上辗转相除法计算次数相对较少.特别当两个数字大小区别较大时计算次数的区别较明显. (2)从结果体现形式来看.辗转相除法体现结果是以相除余数为0则得到.而更相减损术则以减数与差相等而得到.

2.第一次上机操作，实验中遇到问题时，团队协作一起讨论能学到很多知识。自己课后操作时，学会利用网络资源能更加快速高效。这虽然很耗费时间，但是知道了做一个实验的基本步骤和流程。之间分析一个算法的时间复杂性，我喜欢凭直觉，现在知道并学会使用这两种分析时间复杂性的方法。同时也学会如何使用时间函数来对程序计时了。

3.此次实验初步了解了算法分析，三种算法虽然结果相同但是效率却不同。对于复杂度的求解，可以根据我的结果分析得到欧几里得辗转相除算法的是最优算法，欧几里得减法其次，最复杂的是蛮力法。

4.在实验中，对于使用产生随机数测试算法的问题有了新的认识和了解，并学会了使用时间测试函数.从这次实验中，我复习了C++语言代码，同时也通过算法分析用三种方法求解出了最大公约数这个问题。从这个试验的结果我了解到了算法的优与劣的差别，虽然得到的是同样的结果，但是需要的时间和资源却相差很大，这提示我们在以后写算法的时候要找出最优算法。可见算法分析与设计课程的对计编程的人来说是多么的重要，在以后写程序过程中要时刻提醒自己找最优算法，不断在实践中深入认识与学习《算法设计与分析这门课》，同时也感谢赵尔敦老师的悉心指导。

附：核心代码

```c
//求最大公约数的线性组合（扩展欧几里得）
#include<stdio.h>
int x,y;
int gcd(int a,int b)
{                   //欧几里得算法求最大公约数
    if(b==0)
        return a;
    return
        gcd(b,a%b);
}
int exgcd(int a,int b)
{                       //扩展欧几里得算法
    if(b==0)
    {
        x=1;
        y=0;
        return a;
    }
    int r=exgcd(b,a%b);
    int t=x;
    x=y;
    y=t-a/b*y;
    return r;
}
int main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    int d=exgcd(a,b);
    int c=gcd(a,b);
    int k=c/d;
    if(y*k>0)
    printf("%d*%d+%d*%d=%d\n",a,x*k,b,y*k,c);
    else
        printf("%d*%d%d*%d=%d\n",a,x*k,y*k,b,c);
    return 0;
}
```

```
//蛮力法
int CommFactor1(int m,int n){
        int i,factor=1;
        for(i=2;i<=m && i<=n;i++)
        {
            while(m%i==0 && n%i == 0)
            {
                while(m%i==0 && n%i==0)
                {
                    factor=factor*i;
                    m=m/i;
                    n=n/i;
                }
            }
        }
        return factor;
}
```

```
//辗转相除法(非递归)
//有两整数a和b：1 a%b得余数c;2 若c=0，则b即为两数的最大公约数;3 若c≠0，则a=b，b=c，再回去执行1.

int CommFactor2(int a, int b)
{
    int c = a%b;
    while(c != 0)
    {
        a = b;
        b = c;
        c = a%b;
    }
    return b;
}
```

```
/* 更相减损法
有两整数a和b:
    1 若a>b，则a=a-b;
    2 若a<b，则b=b-a;
    3 若a=b，则a（或b）即为两数的最大公约数;
    4 若a≠b，则再回去执行1.
*/
int CommFactor3(int a, int b)
{
    while(a != b)
    {
        if(a > b)
            a = a - b;
        else
            b = b - a;
    }
    return a;
}

```

实验一main.cc ● | 实验一_版本二.h | 实验一_版本三.h | exgcd.c

```
48  {
49      while(a != b)
50      {
51          if(a > b)
52              a = a - b;
53          else
54              b = b - a;
55      }
56      return a;
57  }
58
59  int main()
60  {
61      int a[1000],b[1000];
62      int c,i,j;
63      srand(time(NULL)); /*初始化种子*/
64      for(i=0; i < 1000; i++ ){
65          a[i]=rand()%1001;
66          b[i]=rand()%1001;
67      }
68      printf("%d %d",a[2],a[10]);
69
70      for(i=0; i < 1000; i++){
71          c = CommFactor2(a[i], b[i]);
72      }
73      clock_t start, finish;
74      double duration;
75      start = clock();
76      //c = CommFactor2(a, b);
77      // 穷举法
78      //c = CommFactor3(a, b);
79
80      finish = clock();
81      duration = 10000*(double)(finish-start) / CLOCKS_PER_SEC;
82      printf("elasped time : %f seconds\n", duration);
83      //运行1000次算法，取平均运行时间
84      return 0;
85  }
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
struct timespec
{
 time_t tv_sec;
 long int tv_nsec;
};
int main(void)
{
   struct timespec time_start={0, 0},time_end={0, 0};
   clock_gettime(CLOCK_REALTIME, &time_start);
   printf("start time %llus,%llu ns\n", time_start.tv_sec, time_start.tv_nsec);
   clock_gettime(CLOCK_REALTIME, &time_end);
   printf("end   time %llus,%llu ns\n", time_end.tv_sec, time_end.tv_nsec);
   printf("duration:%llus %llluns\n", time_end.tv_sec-time_start.tv_sec, time_end.tv_nsec-time_start.tv_nsec);
   return 0;
}
```