中山大学数据科学与计算机学院本科生实验报告

（2016 学年春季学期）

课程名称：Algorithm design　　任课教师：张子臻

年级	15	专业（方向）	软件工程(移动信息工程)
学号	15352408	姓名	张镓伟
电话	13531810182	Email	709075442@qq.com
开始日期		完成日期	

目录

1.实验题目

数论专题

Soj 1009 Mersenne Composite N

题意：判断所有质数 i<=k，2^i-1 是否是质数，不是的话就要将它分解质因数输出来。
约束：k<=63

Soj 1020 高精度

题意：题意：输入 n 个整数 bi（1 <= i <= n），以及一个大整数 x，输出一个 n 元组(x mod b1,x mod b2,…,x mod bn)。
约束： n <= 100， 1 < bi <= 1000 (1 <= i <= n) 大整数 x 的位数，m <= 400 并且非负

Soj 1259 Sum of Consecutive Primes

题意：给出一个正整数 n，求出它有多少种方法可以表示成连续的素数的和。 例如 53 = 5 + 7 + 11 + 13 + 17 = 53，共有两种方法。
约束：数字大小 2<=n<=10000

Soj 1240 Faulty Odometer

题意：有个损坏的里程表，不能显示数字 4，会从数字 3 直接跳到数字 5。给出里程表的读数 x，求出实际里程。
约束： 1<=x<=999999999

Soj 1231 The Embarrassed Cryptography

题意：给出两个正整数 K 和 L，问 K 是否存在小于 L 的质因数，有的话则找出最小的质因数。
约束: $4 <= K <= 10^{100}$ ， $2 <= L <= 10^6$

Soj 1203 The Cubic End

题意：题目给出了一个有趣的现象：如果一个数字串，以 1， 3， 7， 9 结尾，则会有一个数，它的三次方以这个数字串结尾，且它自己长度不会超过这个数字串。
约束： 数字串长度 1<=n<=10

Soj 1119 Factstone Benchmark

题意：1960 年发行了 4 位计算机，从此以后每过 10 年，计算机的位数变成两倍。输入某一个年份，求出在这个年份的最大的整数 n 使得 n!能被一个字表示
约束：年份 1960<=n<=2160，且 n%10 == 0

Soj 1500 Prime Gap

题意：给出一个正整数 k，找到与之相邻的两个素数，并求出两个素数之差。如果不存在两个相邻的素数则输出 0。
约束：1<=k<=1299709

2.实验目的

1.学习数论的基础知识，并会做简单的数论题。
2.提高运用课堂所学知识解决实际问题的能力。

3.程序设计

Soj 1009 Mersenne Composite N

解题思路：这题由于只要判断的数其实很少，完全可以打表水过去，但是作为一个退役的 OIer，是不能这么没有出息的。为了解决这道题，我们先知道以下三个知识点：
1.费马小定理：假如 p 是质数，且 gcd(a,p)=1，那么 $a^{(p-1)}\equiv 1$（mod p）。即：假如 a 是整数，p 是质数，且 a,p 互质(即两者只有一个公约数 1)，那么 a 的(p-1)次方除以 p 的余数恒等于 1。这个定理是很常用的，比如我经常用来求 a 在 mop p 下的逆元，就是 $a^{(p-2)}$mod p

2.米勒-拉宾(Miller-Rabin)素数检验。由费马小定理，我们可以猜想出一个验证素数的方法。由于除了 2 之外其他素数都是奇数，那么我们猜测假如数 n 满足 2^(n-1) mod n=1 那 n 不就是素数了吗。但人们已经找到了反例。不过如果不满足 2^(n-1) mod n=1，那么 n 一定为合数这个是可以确定的。米勒-拉宾(Miller-Rabin)素数检验就是为了降低费马小定理检验素数出错的概率而提出的，内容如下：

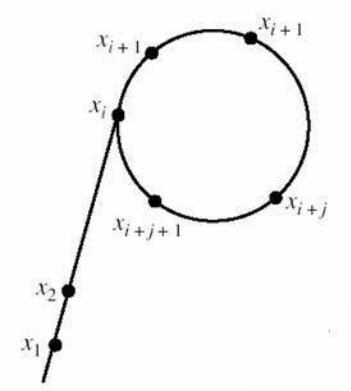
底数由 2 变为 a(a 为小于 n 的任意整数)不断地提取指数 n-1 中的因子 2，把 n-1 表示成 d*2^r（其中 d 是一个奇数）那么我们需要计算的东西就变成了 a 的 d*2^r 次方除以 n 的余数。于是，a^(d * 2^(r-1))%n 要么等于 1，要么等于 n-1。如果 a^(d * 2^(r-1))%n 等于 1，定理继续适用于 a^(d * 2^(r-2))，这样不断开方开下去，直到对于某个 i 满足 a^(d * 2^i) mod n = n-1 或者最后指数中的 2 用完了得到的 a^d mod n=1 或 n-1。这样费马小定理加强成如下的形式：这样，Fermat 小定理加强为如下形式：

尽可能提取因子 2， 把 n-1 表示成 d*2^r，如果 n 是一个素数，那

么或者 a^d mod n=1,或者存在某个 i 使得 a^(d*2^i) mod n=n-1 （ 0<=i<r ）（注意 i 可以等于 0，这就把 a^d mod n=n-1 的情况统一到后面去了）
从上也可看出米勒-拉宾检验是一个概率算法，对于数 n，我们用越多的 a 去检验，正确概率越高。

3. Pollard rho 大整数质因数分解。大数分解最简单的思想也是试除法，就是从 2 到 sqrt(n)，一个一个的试验，直到除到 1 或者循环完，最后判断一下是否已经除到 1 了即可。但是这样的做的复杂度是相当高的。一种很妙的思路是找到一个因子（不一定是质因子），然后再一路分解下去。这就是基于 Miller_rabin 的大数分解法 Pollard_rho 大数分解。这个算法找一个因子是一种随机化算法随机取一个 x1，由 x1 构造 x2，使得｛p 可以整除 x1-x2 && x1-x2 不能整除 n｝则 p=gcd（x1-x2，n），结果可能是 1 也可能不是 1。如果不是 1 就找寻成功了一个因子，返回因子；如果是 1 就寻找失败，那么我们就要不断调整 x2，具体的办法通常是 x2=x2*x2+c（c 是自己定的或者随机也可以）直到 x2 出现了循环==x1 了表示 x1 选取失败重新选取 x1 重复上述过程。

因为 x1 和 x2 再调整时最终一定会出现循环，形成一个类似希腊字母 rho 的形状，故因此得名。



最后回到这题，就是枚举每个质数 i<=k，用米勒-拉宾测试判断 2^i-1 是不是质数，不是的话就用 Pollard_rho 算法分解。
代码：

```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<algorithm>
using namespace std;
#define rep(i,u,v) for (int i=(u);i<=(v);i++)
typedef long long ll;
ll fac[110];
int tot;
int isPrime(int x){
    if (x==2)return 1;
    rep(i,2,(int)sqrt(x))
        if (x%i==0)return 0;
    return 1;
}
//米勒-拉宾素数测试
ll qmul(ll a,ll b,ll c){//快速乘法a*b%c
    a%=c;b%=c;
    ll ans=0;
    while (b){
        if (b&1)ans=(ans+a)%c;
        a=(a<<1)%c;
        b>>=1;
    }
    return ans;
}
```

```
ll qpow(ll a,ll b,ll c){//快速幂a^b % c
    if (b==1)return a%c;
    if (!b)return 1%c;
    ll tmp=qpow(a,b/2,c);
    tmp=qmul(tmp,tmp,c);
    if (b&1)tmp=qmul(tmp,a,c);
    return tmp;
}
int check(ll a,ll d,int r,ll n){
//验证a^(d*2^r) %n=1 或 n-1
    ll tmp=qpow(a,d,n);
    ll last=tmp;
    rep(i,1,r){
        tmp=qmul(tmp,tmp,n);
        if (tmp==1&&last!=1&&last!=n-1)return false;
        last=tmp;
    }
    if (tmp!=1)return false;
    return true;
}
int Miller_Rabin(ll n){
    if ( n<2 || n&1==0 )return 0;
    if (n==2)return 1;
    ll d=n-1; int r=0;
    while ( (d&1)==0 )d>>=1,r++; //将n分解成d*2^r
    rep(i,1,30){//随机测试30次，减小出错概率
        ll a=rand()%(n-1)+1;
        if (!check(a,d,r,n))return false;//不通过测试则是合数
    }
    return true;
}

///Pollard_rho大整数质因数分解
ll gcd(ll a,ll b){
    if (a<0)a=-a;
    if (b<0)b=-b;
    if (!b)return a;
    else return gcd(b,a%b);
}
ll Pollard_rho(ll x,ll c){
    ll i=1,k=2;
    ll x0=rand()%x;
    ll y=x0;
    while(1){
        i++;
        x0=(qmul(x0,x0,x)+c)%x;  //x^2+c
        ll d=gcd(y-x0,x);
        if(d!=1&&d!=x) return d;
        if(y==x0) return x;  //找到循环，算法失败，重来
        if(i==k){y=x0;k+=k;} //将对y的寻找范围扩大2倍，是一个优化
    }
}
void findfac(ll n){
    if (Miller_Rabin(n)){
        fac[++tot]=n;
        return;
    }
    ll p=n;
    while (p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
    findfac(p); //找到了一个因子，那么递归下去继续找
    findfac(n/p);
}

void work(int k){
    ll n=(1LL<<(k))-1;
    tot=0;
    findfac(n);
    if (tot>1){
        sort(fac+1,fac+tot+1);
        printf("%lld",fac[1]);
        rep(i,2,tot) printf(" * %lld",fac[i]);
        printf(" = %lld = ( 2 ^ %d ) - 1\n", n, k);
    }
}
int main(){
    srand(1996-04-22);
    int k;
    scanf("%d",&k);
    rep(i,2,k)
        if (isPrime(i))
            work(i);
    return 0;
}
```

Soj 1020 Big Integer

解题思路：我们知道：(a+b)%m=(a%m+b%m)%m

(a*b)%m=(a%m)*(b%m)%m

而一个 n 位数，假设从高位到低位依次为 $X_{(n-1)}$，$X_{(n-2)}$，…，X_0，则它可以分解为 $X_{(n-1)}*10^{n-1} + X_{(n-2)}*10^{n-2} +\ldots+ X_0*10^0$ 。设 $f[i]= X_{(i)}*10^i$ %m,0<=i<n,那么这题的答案就是

$$(\sum_{i=0}^{n-1} f[i])\%m$$

由观察得每一个位乘上的 10 的某次方是从低位开始一次比一次多乘一个 10，所以这题也可以写成迭代的形式更方便我们写代码：

$g[0]=X_{n-1}\%m$　　$g[i]=(g[i-1]*10+X_{n-1-i})\%m$

最终答案就是 g[n-1].

时间复杂度：O(n)

代码：

```
#include <iostream>
#include <vector>
using namespace std;
#define rep(i,u,v) for (int i=(u);i<=(v);i++)
int calc(char s[], int x){
    int ans=0;
    for (int i=0;s[i];i++){
        ans=(ans*10+(s[i]-'0'))%x;
    }
    return ans;
}

int main(){
    int T,n,x,a[110];
    char s[410];
    cin>>T;
    rep(tt,1,T){
        cin>>n;
        rep(i,1,n)cin>>a[i];
        cin>>s;
        cout<<"(";
        rep(i,1,n){
            int ret=calc(s,a[i]);
            if (i!=n)cout<<ret<<",";
            else cout<<ret<<")\n";
        }
    }
    return 0;
}
```

Soj 1259 Sum of Consecutive Primes

解题思路:要将一个数分成连续一段素数的和，显然我们首先要知道素数有什么。所以我们一开始先快速筛出 10000 以内的所有素数。然后发现只有 1229 个。那么对于给出的每个数 n，我们可以枚举连续一段素数的起点和终点，判断该段之和是否为 n，这一步可以先求出素数的前缀和 sum，那么[i, j]这段的素数和就是 sum[j]-sum[i-1]，这样可以方便不少。最后我们统计有多少段素数满足要求即可。

时间复杂度：O(n^2)

代码:

```cpp
#include<iostream>
using namespace std;
#define rep(i,u,v) for (int i=(u);i<=(v);i++)
int p[1230],v[10001],sum[1230];
int tot;
void init(){
    rep(i,2,10000){
        if (v[i])continue;
        p[++tot]=i;
        sum[tot]=sum[tot-1]+i;
        for (int j=i*i;j<=10000;j+=i)
            v[j]=1;
    }
}
int main(){
    ios::sync_with_stdio(false);
    init();
    int n;
    while (cin>>n&&n){
        int ans=0;
        rep(i,1,tot){
            rep(j,i,tot){
                if (sum[j]-sum[i-1]>n)break;
                if (sum[j]-sum[i-1]==n)ans++;
            }
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

Soj 1240 Faulty Odometer

解题思路:这个损坏的里程表只能显示“012356789”这 9 个数字，它其实就是显示一个 9 进制数。我们将读数转换成一个真的 9 进制数，即判断读数的每一位，如果该位小于 4，则不变，如果大于 4，该位上的数字减一。然后就是简单的九进制转十进制的问题了。假设该 n 位 9 进制数从高位到低位依次为 $X_{(n-1)}$，$X_{(n-2)}$，…，X_0，那么转换成 10 进制的方法就是 $X_{(n-1)}*9^{n-1} + X_{(n-2)}*9^{n-2} +\ldots+ X_0*9^0$

时间复杂度:O(n)

代码:

```cpp
#include<iostream>
using namespace std;
int change(int n){
    int ans=0,pow=1;
    while (n){
        int x=n%10;
        x=x>4?x-1:x;
        ans=ans+pow*x;
        pow*=9;n/=10;
    }
    return ans;
}
int main(){
    ios::sync_with_stdio(false);
    int n;
    while (cin>>n&&n){
        cout<<n<<": "<<change(n)<<endl;
    }
    return 0;
}
```

Soj 1231 The Embarrassed Cryptography

解题思路:首先使用 O(n)线性筛素数的方法筛出 10^6 以内的所有素数。从小到大枚举每一个小于 L 的素数 x，判断能否被 K 整除。但是 K 是个大整数，显然不能直接用 K%x 去判断。将 K 拆成各位数字与 10 的幂之和的形式 $X_{(n-1)}*10^{n-1} + X_{(n-2)}*10^{n-2} +\ldots+ X_0*10^0$。那么 K%x 的结果就相当于这个多项式中每一项%x 之后的和再%x。由于 10 的幂的连续性，我们同样可将这个做法写成一个递推式：

$$g[0]=X_{n-1}\%x \qquad g[i]=(g[i-1]*10+X_{n-1-i})\%m$$

最后只要看 g[n-1]是否为 0，若为 0 则说明 K 能被 x 整除，输出“Bad x”并退出枚举。若到最后都没有 x 能整除 K，则输出“GOOD”

时间复杂度:O(n)

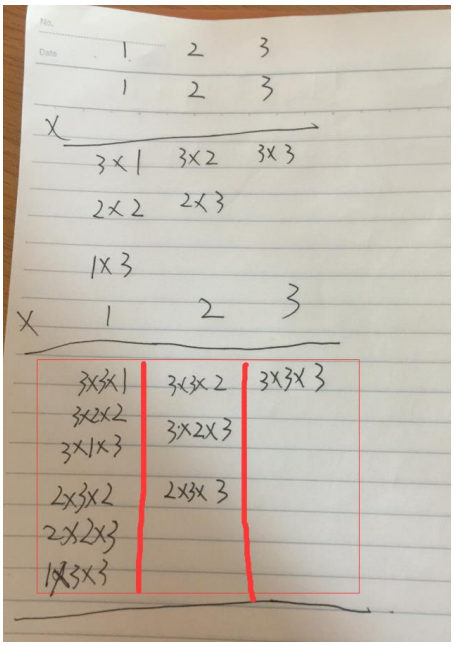
代码:

```
#include<iostream>
#include<cstring>
using namespace std;
#define rep(i,u,v) for (int i=(u);i<=(v);i++)
int p[80000],v[1000001],a[110];
int tot;
void init(){
    rep(i,2,1000000){
        if (!v[i])p[++tot]=i;
        for (int j=1;j<=tot;j++){
            if ((long long)p[j]*i>1000000)break;
            v[p[j]*i]=1;
            if (i%p[j]==0)break;
        }
    }
}
int check(int x){
    int ret=0;
    rep(i,1,a[0])
        ret=(ret*10+a[i])%x;
    return ret!=0;
}

int main(){
    ios::sync_with_stdio(false);
    init();
    int L;
    char K[110];
    while(cin>>K>>L){
        if (K[0]=='0'&&L==0)break;
        a[0]=strlen(K);
        rep(i,0,a[0]-1)a[i+1]=K[i]-'0';
        rep(i,1,tot){
            if (p[i]>=L){
                cout<<"GOOD"<<endl;
                break;
            }
            if (!check(p[i])){
                cout<<"BAD "<<p[i]<<endl;
                break;
            }
            if (i==tot){
                cout<<"GOOD"<<endl;
                break;
            }
        }
    }
    return 0;
}
```

Soj 1203 The Cubic End

解题思路:由于数字串长度 n 很小，只有 10，那么一个明显的想法就是从低位到高位从小到大枚举每一位的数字，再检查是否满足条件。那么现在第一个问题就是如何算出三次方之后某一位的数字是多少。我现在以求 123 的三次方的末三位来做说明：



从图中可以看出，结果的末三位就是红框中对应那一列的数相加。我们现在设 x 表示第一个 123 的第 x 位，设 y 表示第二个 123 的第 y 位，设 z 表示第三个 123 的第 z 位。那么上图整理后第 i 位的结果与 x、y、z 的关系如下表：

i	x	y	z	xyz 对应位乘积
0	0	0	0	3*3*3
1	1	0	0	3*3*2
	0	1	0	3*2*3
	0	0	1	2*3*3
2	2	0	0	3*3*1
	1	1	0	3*2*2
	0	2	0	3*1*3
	1	0	1	2*3*2
	0	1	1	2*2*3
	0	0	2	1*3*3

假设现在枚举到第 i 位(最低位为第 0 位)，数字为 b[i]，那么结合上表我们不难发现，结果的第 i 位数字，就是所有的 b[x]*b[y]*b[z]之和再加上低位的进位之后再%10，其中（x+y+z=i）。

有了上述结论，我们只有从小到大枚举每一位的数字，判断是否符合要

求，一旦某位找到一个数字符合要求，就退出当前位的枚举而转向下一位数字的枚举，从而保证最后求出的数字是最小的。

时间复杂度:O(n^3)

代码:

```
#include<iostream>
#include<cstring>
using namespace std;
#define rep(i,u,v) for (int i=(u);i<=(v);i++)
#define dto(i,u,v) for (int i=(u);i>=(v);i--)
int a[15],b[15];

int main(){
    ios::sync_with_stdio(false);
    int T,n;
    char s[15];
    cin>>T;
    rep(tt,1,T){
        cin>>s;
        n=strlen(s);
        rep(i,0,n-1)a[i]=s[n-1-i]-'0';
        int carry=0;
        rep(i,0,n-1){
            rep(j,0,9){
                int tmp=carry;
                b[i]=j;
                rep(x,0,i)
                    rep(y,0,i){
                        if (x+y>i)break;
                        int z=i-x-y;
                        tmp+=b[x]*b[y]*b[z];
                    }
                if (tmp%10==a[i]){
                    carry=tmp/10;
                    break;
                }
            }
        }
        while (b[n-1]==0&&n-1>0)n--;
        dto(i,n-1,0)cout<<b[i];
        cout<<endl;
    }
    return 0;
}
```

Soj 1119 Factstone Benchmark

解题思路:这题很有意思。计算机一开始有 4 位，每 10 年位数乘 2。也就是说第 1960<=x<=2160 年的位数 len 就是 $4*2^{(x-1960)/10}=2^{(x-1960/10)+2}$。题目要我们求一个最大的 n 使得 n!可以在第 x 年用 len 位表示。而 len 位可以表示的范围为$[0,2^{len}-1]$也就是有：

$$n!<2^{len}$$

显然如果我们想算出 n！与 2^len 比较是不可行的，因为容易算出 len 最大为 2^22，而 2^(2^22)太大了。但是如果我们发现虽然 2^len 比较大，但是 len 还是在可接受范围内，因为 2^22 是四百多万。所以我们对上式两边取对数得

$$\log_2(n!) < len$$

由于 C++标准库中没有以 2 为底的对数函数只有以 10 为底的对数函数 log，所以我们的等式再变形为：

$$lg(n!)/lg(2)<len$$
$$lg(n!)<lg(2)*len$$

$lg(1)+lg(2)+\cdots+lg(n)<lg(2)*len$

如此就转换成了求一个最大前缀和不超过一个值的简单问题了。只要从小到大枚举 n 去求即可。

时间复杂度：O(n)

代码:

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
    int n;
    while (cin>>n&&n){
        int num=2+(n-1960)/10;
        int len=1;
        for (int i=1;i<=num;i++)len*=2;
        double sum=log(1),bit=log(2)*len;
        for (int i=2;;i++){
            if (sum+log(i)>=bit){
                cout<<i-1<<endl;
                break;
            }
            sum+=log(i);
        }
    }
    return 0;
}
```

Soj 1500 Prime Gap

解题思路:由于数的范围达到百万级别。所以我们选择先使用线性筛素数的方法筛出 1-1299709 之内的所有素数，共 100000 个。对给定的每个数，遍历素数数组找到其相邻的两个素数求得它们的差就好。要注意小于数组开头素数和大于数组末尾素数的数以及本身就是素数的数就输出 0.

时间复杂度：O(n)

代码:

```
#include<iostream>
using namespace std;
#define rep(i,u,v) for (int i=(u);i<=(v);i++)
int p[100002],v[1300001];
int tot;
void init(){
    rep(i,2,1299709){
        if (!v[i])p[++tot]=i;
        for (int j=1;j<=tot;j++){
            if ((long long)p[j]*i>1299709)break;
            v[p[j]*i]=1;
            if (i%p[j]==0)break;
        }
    }
}
int main(){
    ios::sync_with_stdio(false);
    init();
    int n,ans;
    while(cin>>n&&n){
        ans=0;
        rep(i,1,tot){
            if (p[i]==n)break;
            if (p[i]<n&&i==tot)break;
            if (p[i]>n)break;
            if (p[i]<n&&p[i+1]>n){
                ans=p[i+1]-p[i];
                break;
            }
        }
```

```
31          cout<<ans<<endl;
32      }
33      return 0;
34  }
```

4.程序运行与测试

8 道题均通过成功运行并通过 sicily 的测试。

5118823		smie15352408	1009	C++	Accepted	0.02sec	308 KB	2382 Bytes	2017-05-18 16:31:14
5118209		smie15352408	1203	C++	Accepted	0sec	300 KB	921 Bytes	2017-05-18 11:47:45
5117553		smie15352408	1119	C++	Accepted	0.18sec	308 KB	419 Bytes	2017-05-17 23:40:48
5117502		smie15352408	1231	C++	Accepted	0.78sec	4520 KB	1036 Bytes	2017-05-17 23:24:14
5117479		smie15352408	1231	C++	Accepted	0.9sec	4520 KB	1102 Bytes	2017-05-17 23:08:38
5117438		smie15352408	1500	C++	Accepted	0.19sec	5768 KB	574 Bytes	2017-05-17 22:40:45
5117410		smie15352408	1240	C++	Accepted	0.2sec	300 KB	345 Bytes	2017-05-17 22:20:36
5117385		smie15352408	1259	C++	Accepted	0sec	348 KB	624 Bytes	2017-05-17 22:03:43
5117152		smie15352408	1020	C++	Accepted	0.03sec	308 KB	552 Bytes	2017-05-17 19:48:29

5.实验总结与心得

这次数论专题仅是接触数论中很基本的知识，题目基本上都与质数有关。在这里我复习了一下如何快速线性筛素数。对于 1009 这道题，还顺便复习了一下费马小定理以及米勒-拉宾素数测试算法，这两者还是很好用的。但是至于 Pollard_rho 大整数质因数分解这个算法则是我第一次接触。它其实是采用了生日悖论去寻找因素。我们知道在[1,1000]中随机取一个数取得 x 的概率为 1/1000，但是如果随机取两个数 i 和 j，它们的差值为 x 的概率就要大一些。而如果我们从中选取 k 个数，这 k 个数中两两差值为 x 的概率显然又比在[1,1000]中取两个差值为 x 的数的概率要高。这其实就是生日问题，相当于在一堆人中选两个人，他们生日相同的概率是多少。Pollard_rho 算法不需要盲目地找这样的两个数，只需要一个一个地生成并检查梁旭的两个知道找到我们想要的。在这里又不得不感叹前人真是聪明。

附录、提交文件清单

实验报告.pdf
1009.cpp
1020.cpp
1259.cpp
1240.cpp
1231.cpp
1203.cpp
1119.cpp

1500.cpp