# 質數建表與質數測試

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

http://www.csie.ntnu.edu.tw/~u91029/Prime.html

Pei-yih Ting

# 應用題目

- UVa 294 406 516 524 543 583 686 897 914 10140 10168 10179 10290 10299 10311 10329 10392 10394 10490 10622 10650 10780 10791 10852 10820 10879 10948 10956 10990 11064 11327 11408 11424 11426 11466 11476 11752 11889 12005 12493
- ICPC 2116
- PKU 1811

# 質數相關的定義和特性

- 只能被1和它自己整除的就是質數

# 質數相關的定義和特性

- 只能被1和它自己整除的就是質數

  不是質數的就是合成數

# 質數相關的定義和特性

- 只能被1和它自己整除的就是質數

  不是質數的就是合成數

- 質數有無窮多個 – Gauss

# 質數相關的定義和特性

- 只能被1和它自己整除的就是質數

  不是質數的就是合成數

- 質數有無窮多個 – Gauss
  - 人類知道的最大質數是? by GIMP (prime95)

    (2019/01) $2^{82,589,933} - 1$， 十進位 24,862,048 位數

# 質數相關的定義和特性

- 只能被1和它自己整除的就是質數

  不是質數的就是合成數

- 質數有無窮多個 – Gauss
  - 人類知道的最大質數是? by GIMP (prime95)

    (2019/01)　$2^{82,589,933} - 1$，十進位 24,862,048 位數
- 質數定理 (Prime Number Theorem) – Gauss

# 質數相關的定義和特性

- 只能被1和它自己整除的就是質數

  不是質數的就是合成數

- 質數有無窮多個 – Gauss
  - 人類知道的最大質數是? by GIMP (prime95)

    (2019/01)　$2^{82,589,933} - 1$，十進位 24,862,048 位數
- 質數定理 (Prime Number Theorem) – Gauss
  - 質數個數　$\pi(n) \approx \frac{n}{\log n}$

# 質數相關的定義和特性

- 只能被1和它自己整除的就是質數

  不是質數的就是合成數

- 質數有無窮多個 – Gauss
  - 人類知道的最大質數是? by GIMP (prime95)

    (2019/01) $2^{82,589,933} - 1$， 十進位 24,862,048 位數
- 質數定理 (Prime Number Theorem) – Gauss
  - 質數個數 $\pi(n) \approx \frac{n}{\log n}$
  - 質數的密度? 十進位 1000 位數 ≈ 1/2300

    100 位數 ≈ 1/250

    $2^{31}$-1 ($10^9$) 裡有 105,097,565 個 ≈ 1/20

# 相關的定義和特性 (續)

- 證明質數

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數 – 因數

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數 – 因數 或是 費瑪小定理 (Fermat's Little Theorem)

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數 – 因數 或是 費瑪小定理 (Fermat's Little Theorem)
- 質數/合成數的判斷

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數 – 因數 或是 費瑪小定理 (Fermat's Little Theorem)
- 質數/合成數的判斷
  - 沒有效率: Trial division, Lucas test, Agrawal-Kayal-Saxena test

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數 – 因數 或是 費瑪小定理 (Fermat's Little Theorem)
- 質數/合成數的判斷
  - 沒有效率: Trial division, Lucas test, Agrawal-Kayal-Saxena test
  - 或是機率式: Fermat test, Miller-Rabin test

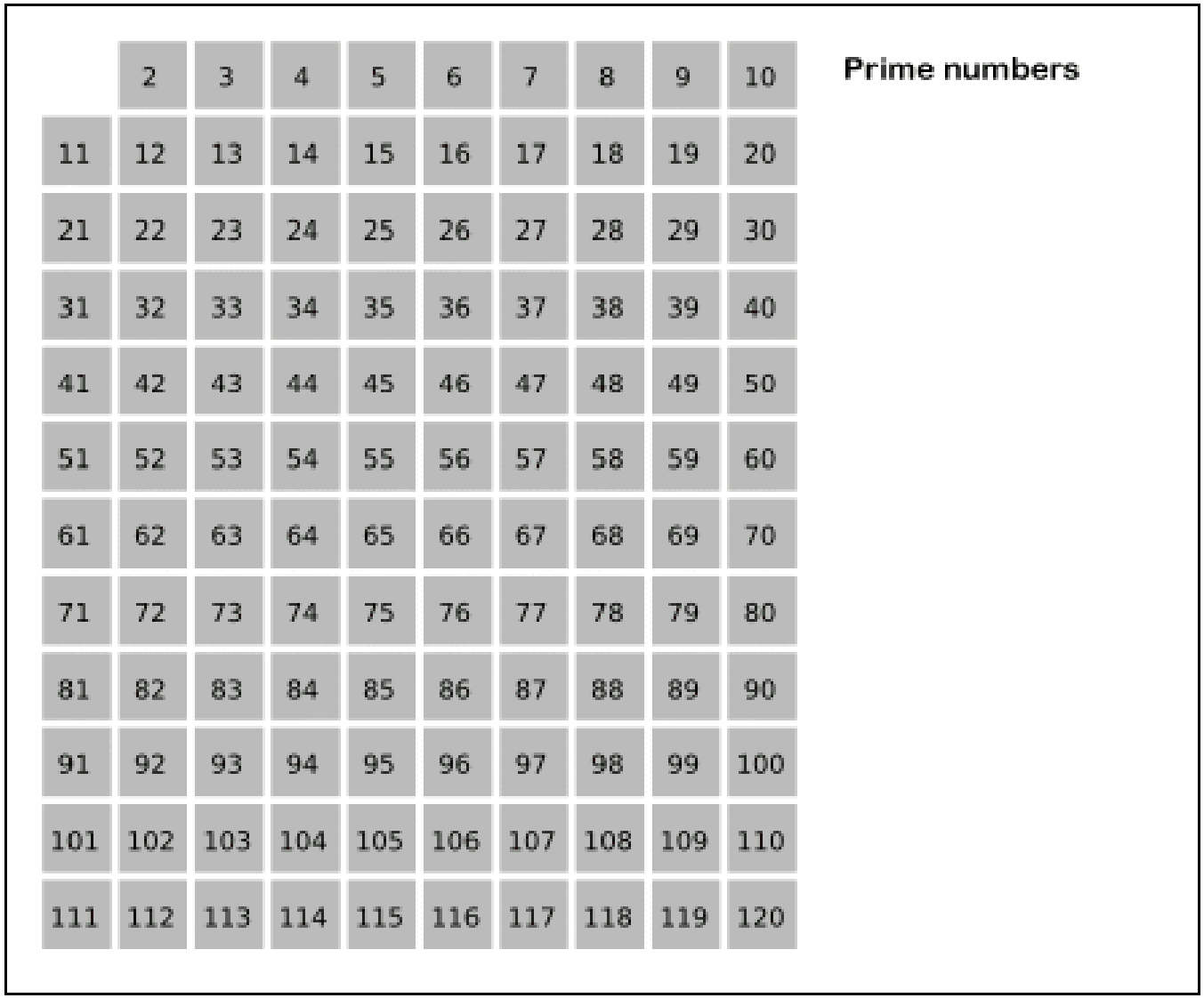
# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數 – 因數 或是 費瑪小定理 (Fermat's Little Theorem)
- 質數/合成數的判斷
  - 沒有效率: Trial division, Lucas test, Agrawal-Kayal-Saxena test
  - 或是機率式: Fermat test, Miller-Rabin test
- 還好除了密碼學/資訊安全的應用之外，大部分需要質數的程式問題裡質數的範圍都在 $2^{32}$ 以下

# 相關的定義和特性 (續)

- 證明質數
  - 蛤, 有效率地證明一個數字是質數?　沒有 1 和 n 以外的因數?
  - 當然不是　　Pratt's Primality Certificate – polynomial length
- 證明合成數 – 因數 或是 費瑪小定理 (Fermat's Little Theorem)
- 質數/合成數的判斷
  - 沒有效率: Trial division, Lucas test, Agrawal-Kayal-Saxena test
  - 或是機率式: Fermat test, Miller-Rabin test
- 還好除了密碼學/資訊安全的應用之外，大部分需要質數的程式問題裡質數的範圍都在 $2^{32}$ 以下
  - 暴力表列法: **Eratosthenes 法**

# Eratosthenes 篩選法




| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |
| 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 |
| 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 |

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {




}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;




}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)




}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數




}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數



}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數
         for (j=i+i; j<=MAX; j+=i) prime[j] = 0; // 將其倍數標示為非質數


}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數
         for (j=i+i; j<=MAX; j+=i) prime[j] = 0; // 將其倍數標示為非質數
   for (i=j=0; i<=MAX; i++) j += prime[i];

}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數
         for (j=i+i; j<=MAX; j+=i) prime[j] = 0; // 將其倍數標示為非質數
   for (i=j=0; i<=MAX; i++) j += prime[i];
   printf("%d", j); // 區間 [1,200000000] 裡共有 11,078,937 個質數
}
```

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數
         for (j=i+i; j<=MAX; j+=i) prime[j] = 0; // 將其倍數標示為非質數
   for (i=j=0; i<=MAX; i++) j += prime[i];
   printf("%d", j); // 區間 [1,200000000] 裡共有 11,078,937 個質數
}
```

記憶體用得有點多

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數
         for (j=i+i; j<=MAX; j+=i) prime[j] = 0; // 將其倍數標示為非質數
   for (i=j=0; i<=MAX; i++) j += prime[i];
   printf("%d", j); // 區間 [1,200000000] 裡共有 11,078,937 個質數
}
```

記憶體用得有點多　> 64MB

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數
         for (j=i+i; j<=MAX; j+=i) prime[j] = 0; // 將其倍數標示為非質數
   for (i=j=0; i<=MAX; i++) j += prime[i];
   printf("%d", j); // 區間 [1,200000000] 裡共有 11,078,937 個質數
}
```

記憶體用得有點多 > 64MB
有點慢

# 程式實作 I

- 實作請參考 演算法筆記
  http://www.csie.ntnu.edu.tw/~u91029/Prime.html

```
const int MAX=200000000;  // 2 x 10^8
char prime[MAX+1]; // 200MB
void eratosthenes() {
   int i, j;
   for (i=0; i<=MAX; i++) prime[i] = 1; // 初始化 (假設都是質數)
   prime[0] = prime[1] = 0;   // 0 和 1 不是質數
   for (i=2; i<=MAX; i++)  // 找下一個標示為質數的數字
      if (prime[i]) // 如果 i 為質數
         for (j=i+i; j<=MAX; j+=i) prime[j] = 0; // 將其倍數標示為非質數
   for (i=j=0; i<=MAX; i++) j += prime[i];
   printf("%d", j); // 區間 [1,200000000] 裡共有 11,078,937 個質數
}
```

記憶體用得有點多　> 64MB

有點慢　需要 5~10 秒 > 1 秒

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈
  - 第一是換成 string.h 裡面的 memset()，很多 OJ 機器上這個函式是最佳化過的，有機會比較快一些

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈
  - 第一是換成 string.h 裡面的 memset()，很多 OJ 機器上這個函式是最佳化過的，有機會比較快一些
  - 第二是當 MAX 不是質數時，一定有質因數會 ≤ sqrt(MAX)

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈
  - 第一是換成 string.h 裡面的 memset()，很多 OJ 機器上這個函式是最佳化過的，有機會比較快一些
  - 第二是當 MAX 不是質數時，一定有質因數會 ≤ sqrt(MAX)

```
memset(prime, 1, sizeof(prime)); // 初始化 (假設都是質數)
```

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈
  - 第一是換成 string.h 裡面的 memset()，很多 OJ 機器上這個函式是最佳化過的，有機會比較快一些
  - 第二是當 MAX 不是質數時，一定有質因數會 ≤ sqrt(MAX)

```
memset(prime, 1, sizeof(prime)); // 初始化 (假設都是質數)
prime[0] = prime[1] = 0;   // 0 和 1 不是質數
```

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈
  - 第一是換成 string.h 裡面的 memset()，很多 OJ 機器上這個函式是最佳化過的，有機會比較快一些
  - 第二是當 MAX 不是質數時，一定有質因數會 ≤ sqrt(MAX)

```
memset(prime, 1, sizeof(prime)); // 初始化 (假設都是質數)
prime[0] = prime[1] = 0;   // 0 和 1 不是質數
int sqrt_root = sqrt((double)MAX);
```

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈
  - 第一是換成 string.h 裡面的 memset()，很多 OJ 機器上這個函式是最佳化過的，有機會比較快一些
  - 第二是當 MAX 不是質數時，一定有質因數會 ≤ sqrt(MAX)

```
memset(prime, 1, sizeof(prime)); // 初始化 (假設都是質數)
prime[0] = prime[1] = 0;   // 0 和 1 不是質數
int sqrt_root = sqrt((double)MAX);
for (i=2; i<=sqrt_root; i++) // 找下一個標示為質數的數字
   if (prime[i]) // 如果 i 為質數
      for (j=i+i; j<=MAX; j+=i) prime[j] = 0;
```

# 程式實作 II

- 建表時間在 i7-3770 上最佳化以後要 4.5 秒，才到 $2*10^8$ 而已
- 如果要到 $2*10^9$ ($2^{31}$)，而且大部分 OJ 的速度沒有很快，時間限制常常都在 3 秒左右，這個速度不太能夠接受
- 速度的問題主要是在於迴圈
  - 第一是換成 string.h 裡面的 memset()，很多 OJ 機器上這個函式是最佳化過的，有機會比較快一些
  - 第二是當 MAX 不是質數時，一定有質因數會 ≤ sqrt(MAX)

```
memset(prime, 1, sizeof(prime)); // 初始化 (假設都是質數)
prime[0] = prime[1] = 0;   // 0 和 1 不是質數
int sqrt_root = sqrt((double)MAX);
for (i=2; i<=sqrt_root; i++) // 找下一個標示為質數的數字
   if (prime[i]) // 如果 i 為質數
      for (j=i+i; j<=MAX; j+=i) prime[j] = 0;
```

變成 **2.8** 秒了

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?
  - 除了 2 和 3 之外，所有的質數都是 6k-1 或是 6k+1 這種型式耶!

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?
  - 除了 2 和 3 之外，所有的質數都是 6k-1 或是 6k+1 這種型式耶!
  - 原本內層迴圈由每一個質數 i 的 2 倍開始測試，其實從它的平方 i*i 開始測就可以，因為小於 i*i 的數字，如果一個因數是 i，另外一個因數一定是小於 i 的，已經被前面的迴圈測試過了

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?
  - 除了 2 和 3 之外，所有的質數都是 6k-1 或是 6k+1 這種型式耶!
  - 原本內層迴圈由每一個質數 i 的 2 倍開始測試，其實從它的平方 i*i 開始測就可以，因為小於 i*i 的數字，如果一個因數是 i，另外一個因數一定是小於 i 的，已經被前面的迴圈測試過了

```
for (i=2*2; i<=MAX; i+=2) prime[j] = 0;
for (i=3*3; i<=MAX; i+=3) prime[j] = 0;
```

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?
  - 除了 2 和 3 之外，所有的質數都是 6k-1 或是 6k+1 這種型式耶!
  - 原本內層迴圈由每一個質數 i 的 2 倍開始測試，其實從它的平方 i*i 開始測就可以，因為小於 i*i 的數字，如果一個因數是 i，另外一個因數一定是小於 i 的，已經被前面的迴圈測試過了

```
for (i=2*2; i<=MAX; i+=2) prime[j] = 0;
for (i=3*3; i<=MAX; i+=3) prime[j] = 0;
for (i=6; i<=sqrt_value; i+=6)  { // 除了 2, 3 其它質數都是 6n+1/6n-1



}
```

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?
  - 除了 2 和 3 之外，所有的質數都是 6k-1 或是 6k+1 這種型式耶!
  - 原本內層迴圈由每一個質數 i 的 2 倍開始測試，其實從它的平方 i*i 開始測就可以，因為小於 i*i 的數字，如果一個因數是 i，另外一個因數一定是小於 i 的，已經被前面的迴圈測試過了

```
for (i=2*2; i<=MAX; i+=2) prime[j] = 0;
for (i=3*3; i<=MAX; i+=3) prime[j] = 0;
for (i=6; i<=sqrt_value; i+=6)  { // 除了 2, 3 其它質數都是 6n+1/6n-1
   if (prime[k=i-1])
      for (j=k*k; j<=MAX; j+=k) prime[j] = 0;


}
```

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?
  - 除了 2 和 3 之外，所有的質數都是 6k-1 或是 6k+1 這種型式耶!
  - 原本內層迴圈由每一個質數 i 的 2 倍開始測試，其實從它的平方 i*i 開始測就可以，因為小於 i*i 的數字，如果一個因數是 i，另外一個因數一定是小於 i 的，已經被前面的迴圈測試過了

```
for (i=2*2; i<=MAX; i+=2) prime[j] = 0;
for (i=3*3; i<=MAX; i+=3) prime[j] = 0;
for (i=6; i<=sqrt_value; i+=6) { // 除了 2, 3 其它質數都是 6n+1/6n-1
   if (prime[k=i-1])
      for (j=k*k; j<=MAX; j+=k) prime[j] = 0;
   if (prime[k=i+1])
      for (j=k*k; j<=MAX; j+=k) prime[j] = 0;
}
```

# 程式實作 III

- 建質數表就已經花掉那麼多時間，這樣子就沒有什麼時間處理其它的問題了，還需要再加快一點速度才好
  - 想要再減少一點迴圈的次數，何必由 2 開始一個一個數字測試呢?
  - 偶數直接跳掉會不會少一點點時間呢?
  - 除了 2 和 3 之外，所有的質數都是 6k-1 或是 6k+1 這種型式耶!
  - 原本內層迴圈由每一個質數 i 的 2 倍開始測試，其實從它的平方 i*i 開始測就可以，因為小於 i*i 的數字，如果一個因數是 i，另外一個因數一定是小於 i 的，已經被前面的迴圈測試過了

```
for (i=2*2; i<=MAX; i+=2) prime[j] = 0;
for (i=3*3; i<=MAX; i+=3) prime[j] = 0;
for (i=6; i<=sqrt_value; i+=6) { // 除了 2, 3 其它質數都是 6n+1/6n-1
   if (prime[k=i-1])
      for (j=k*k; j<=MAX; j+=k) prime[j] = 0;
   if (prime[k=i+1])
      for (j=k*k; j<=MAX; j+=k) prime[j] = 0;
}
```

沒有什麼進展，
只稍微快一點
**2.6** 秒了

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}-1$ 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}-1$ 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**
- 下面程式裡用位元 and 00100000 (& mask[5])運算來擷取第 **5** 位元出來檢查，用位元 and 11011111 (& imask[5]) 運算來清除第 **5** 位元

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**
- 下面程式裡用位元 and 00100000 (& mask[5])運算來擷取第 **5** 位元出來檢查，用位元 and 11011111 (& imask[5]) 運算來清除第 **5** 位元

```
unsigned char prime[MAX/8+1]; // 269MB  268435456
unsigned char masks[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
unsigned char imasks[] = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
```

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**
- 下面程式裡用位元 and 00100000 (& mask[5])運算來擷取第 **5** 位元出來檢查，用位元 and 11011111 (& imask[5]) 運算來清除第 **5** 位元

```
unsigned char prime[MAX/8+1]; // 269MB  268435456
unsigned char masks[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
unsigned char imasks[] = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
for (i=2*2; i<=MAX; i+=2) prime[i/8] &= imasks[i%8];
for (i=3*3; i<=MAX; i+=3) prime[i/8] &= imasks[i%8];
```

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**
- 下面程式裡用位元 and 00100000 (& mask[5])運算來擷取第 **5** 位元出來檢查，用位元 and 11011111 (& imask[5]) 運算來清除第 **5** 位元

```
unsigned char prime[MAX/8+1]; // 269MB  268435456
unsigned char masks[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
unsigned char imasks[] = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
for (i=2*2; i<=MAX; i+=2) prime[i/8] &= imasks[i%8];
for (i=3*3; i<=MAX; i+=3) prime[i/8] &= imasks[i%8];
for (i=6; i<=sqrt_value; i+=6) {


}
```

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**
- 下面程式裡用位元 and 00100000 (& mask[5])運算來擷取第 **5** 位元出來檢查，用位元 and 11011111 (& imask[5]) 運算來清除第 **5** 位元

```
unsigned char prime[MAX/8+1]; // 269MB  268435456
unsigned char masks[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
unsigned char imasks[] = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
for (i=2*2; i<=MAX; i+=2) prime[i/8] &= imasks[i%8];
for (i=3*3; i<=MAX; i+=3) prime[i/8] &= imasks[i%8];
for (i=6; i<=sqrt_value; i+=6) {
        if (k=i-1, prime[k/8]&masks[k%8])        if (k=i+1, prime[k/8]&masks[k%8])
            for (j=k*k; j<=MAX; j+=k)                for (j=k*k; j<=MAX; j+=k)
                prime[j/8] &= imasks[j%8];               prime[j/8] &= imasks[j%8];
}
```

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**
- 下面程式裡用位元 and 00100000 (& mask[5])運算來擷取第 **5** 位元出來檢查，用位元 and 11011111 (& imask[5]) 運算來清除第 **5** 位元

```
unsigned char prime[MAX/8+1]; // 269MB  268435456
unsigned char masks[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
unsigned char imasks[] = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
for (i=2*2; i<=MAX; i+=2) prime[i/8] &= imasks[i%8];
for (i=3*3; i<=MAX; i+=3) prime[i/8] &= imasks[i%8];
for (i=6; i<=sqrt_value; i+=6) {
        if (k=i-1, prime[k/8]&masks[k%8])
            for (j=k*k; j<=MAX; j+=k)
                prime[j/8] &= imasks[j%8];
        if (k=i+1, prime[k/8]&masks[k%8])
            for (j=k*k; j<=MAX; j+=k)
                prime[j/8] &= imasks[j%8];
}
```

用了 **18.2** 秒

# 程式實作 IV

- 時間上沒有太多進展了，目前程式能夠建表到 $2^{31}$-1 嗎?
- 目前用一個位元組標示一個數字是不是質數，如果要作到 $2^{31}$，需要十倍的記憶體，就是2GB了，現在程式的環境裡要拿到 2GB 的記憶體需要動態配置，也幾乎是上限了，靜態的全域陣列大概可以用到1GB，一般 OJ 的環境常常限制 64MB，差異蠻大的，一定得縮小一點
- 讓我們用一個位元來標示一個數字是不是質數，這樣子只需要**269MB**
- 下面程式裡用位元 and 00100000 (& mask[5])運算來擷取第 **5** 位元出來檢查，用位元 and 11011111 (& imask[5]) 運算來清除第 **5** 位元

```
unsigned char prime[MAX/8+1]; // 269MB  268435456
unsigned char masks[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
unsigned char imasks[] = {0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F};
for (i=2*2; i<=MAX; i+=2) prime[i/8] &= imasks[i%8];
for (i=3*3; i<=MAX; i+=3) prime[i/8] &= imasks[i%8];
for (i=6; i<=sqrt_value; i+=6) {
        if (k=i-1, prime[k/8]&masks[k%8])
            for (j=k*k; j<=MAX; j+=k)
                prime[j/8] &= imasks[j%8];
        if (k=i+1, prime[k/8]&masks[k%8])
            for (j=k*k; j<=MAX; j+=k)
                prime[j/8] &= imasks[j%8];
}
```

用了 **18.2** 秒
共有 **105,097,565** 個質數

# 程式實作 V

- 記憶體還可以再少用一點嗎?

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊?

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

  除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

prime[i/16]&masks[i/2%8]

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

unsigned char prime[MAX/**16**+1]; // **128MB** 134217729 only odd numbers

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

unsigned char prime[MAX/**16**+1]; // **128MB** 134217729 only odd numbers
prime[1/16] &= imasks[1/2];

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

```
unsigned char prime[MAX/16+1];  // 128MB  134217729 only odd numbers
prime[1/16] &= imasks[1/2];
for (i=3*3; i<=MAX; i+=6) prime[i/16] &= imasks[(i/2)%8];
```

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

```
unsigned char prime[MAX/16+1];  // 128MB  134217729 only odd numbers
prime[1/16] &= imasks[1/2];
for (i=3*3; i<=MAX; i+=6) prime[i/16] &= imasks[(i/2)%8];
int sqrt_value = sqrt((double)MAX) + 1; // 46340.95 + 1
for (i=6; i<=sqrt_value; i+=6) {




}
```

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

```
unsigned char prime[MAX/16+1];  // 128MB  134217729 only odd numbers
prime[1/16] &= imasks[1/2];
for (i=3*3; i<=MAX; i+=6) prime[i/16] &= imasks[(i/2)%8];
int sqrt_value = sqrt((double)MAX) + 1; // 46340.95 + 1
for (i=6; i<=sqrt_value; i+=6) {
   if (k=i-1, prime[k/16]&masks[(k/2)%8])
        for (j=k*k; j<=MAX; j+=k+k) prime[j/16] &= imasks[(j/2)%8];


}
```

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

```
unsigned char prime[MAX/16+1];  // 128MB  134217729 only odd numbers
prime[1/16] &= imasks[1/2];
for (i=3*3; i<=MAX; i+=6) prime[i/16] &= imasks[(i/2)%8];
int sqrt_value = sqrt((double)MAX) + 1; // 46340.95 + 1
for (i=6; i<=sqrt_value; i+=6) {
    if (k=i-1, prime[k/16]&masks[(k/2)%8])
        for (j=k*k; j<=MAX; j+=k+k) prime[j/16] &= imasks[(j/2)%8];
    if (k=i+1, prime[k/16]&masks[(k/2)%8])
        for (j=k*k; j<=MAX; j+=k+k) prime[j/16] &= imasks[(j/2)%8];
}
```

# 程式實作 V

- 記憶體還可以再少用一點嗎?
- 偶數何必也用一個位元標示啊? 省下一半的記憶體 128MB!!!

除了 2 其它質數都是奇數, prime[] 陣列裡只標示奇數

| | i | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/2%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/16]&masks[i/2%8] | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | … |

```
unsigned char prime[MAX/16+1];  // 128MB  134217729 only odd numbers
prime[1/16] &= imasks[1/2];
for (i=3*3; i<=MAX; i+=6) prime[i/16] &= imasks[(i/2)%8];
int sqrt_value = sqrt((double)MAX) + 1; // 46340.95 + 1
for (i=6; i<=sqrt_value; i+=6) {
    if (k=i-1, prime[k/16]&masks[(k/2)%8])
        for (j=k*k; j<=MAX; j+=k+k) prime[j/16] &= imasks[(j/2)%8];
    if (k=i+1, prime[k/16]&masks[(k/2)%8])
        for (j=k*k; j<=MAX; j+=k+k) prime[j/16] &= imasks[(j/2)%8];
}
```

只用了 **9.73** 秒

# 程式實作 VI

- 記憶體再少用一點

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| | i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| | i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| | i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |

prime[i/24]&masks[i/3%8]

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| i | | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/24]&masks[i/3%8] | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | … |

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| | i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/24]&masks[i/3%8] | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | … |

unsigned char prime[MAX/**24**+1]; // **85MB** 89478485 only 6k+1/6k-1 numbers

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| | i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/24]&masks[i/3%8] | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | … |

unsigned char prime[MAX/**24**+1]; // **85MB** 89478485 only 6k+1/6k-1 numbers
prime[1/24] &= imasks[(1/3)%8];

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| | i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/24]&masks[i/3%8] | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | … |

```
unsigned char prime[MAX/24+1];  // 85MB  89478485 only 6k+1/6k-1 numbers
prime[1/24] &= imasks[(1/3)%8];
int sqrt_value = sqrt((double)MAX) + 1; // 46340.95 + 1
for (i=6; i<=sqrt_value; i+=6) {




}
```

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| | i | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/24]&masks[i/3%8] | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | … |

```
unsigned char prime[MAX/24+1];  // 85MB  89478485 only 6k+1/6k-1 numbers
prime[1/24] &= imasks[(1/3)%8];
int sqrt_value = sqrt((double)MAX) + 1; // 46340.95 + 1
for (i=6; i<=sqrt_value; i+=6) {
    if (k=i-1, prime[k/24]&masks[(k/3)%8])
        for (j=k*k,z=2; j<=MAX; j+=z*k,z=6-z)    // (6k-1)*(6k-1)=6(6k-2)k+1
            prime[j/24] &= imasks[(j/3)%8];      // +2k, +4k, +2k, +4k, …



}
```

# 程式實作 VI

- 記憶體再少用一點
- 除了 2 和 3 之外，只有 **6k-1** 和 **6k+1** 可以是質數

| i | | 1 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 25 | 29 | 31 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| byte | i/24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | … |
| bit | i/3%8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | … |
| prime[i/24]&masks[i/3%8] | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | … |

```
unsigned char prime[MAX/24+1];  // 85MB  89478485 only 6k+1/6k-1 numbers
prime[1/24] &= imasks[(1/3)%8];
int sqrt_value = sqrt((double)MAX) + 1; // 46340.95 + 1
for (i=6; i<=sqrt_value; i+=6) {
    if (k=i-1, prime[k/24]&masks[(k/3)%8])
        for (j=k*k,z=2; j<=MAX; j+=z*k,z=6-z)    // (6k-1)*(6k-1)=6(6k-2)k+1
            prime[j/24] &= imasks[(j/3)%8];      // +2k, +4k, +2k, +4k, …
    if (k=i+1, prime[k/24]&masks[(k/3)%8])
        for (j=k*k,z=4; j<=MAX; j+=z*k,z=6-z)    // (6k+1)*(6k+1)=6(6k+2)k+1
            prime[j/24] &= imasks[(j/3)%8];      // +4k, +2k, +4k, +2k, …
}
```

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

5 7 11 13 17 19 23 25 29 31 …

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

5 7 11 13 17 19 23 25 29 31 …
+2 +4 +2 +4 +2 +4 +2 +4 +2 …

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

5 7 11 13 17 19 23 25 29 31 …
+2 +4 +2 +4 +2 +4 +2 +4 +2 …

```
for (i=5,z1=2; i<=sqrt_value; i+=z1,z1=6-z1)
  if (isprime[i/24]&masks[(i/3)%8])
```

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

```
5   7   11  13  17  19  23  25  29  31  …
  +2  +4  +2  +4  +2  +4  +2  +4  +2   …
```

```
for (i=5,z1=2; i<=sqrt_value; i+=z1,z1=6-z1)
  if (isprime[i/24]&masks[(i/3)%8])
    for (j=i*i,z2=z1; j<=MAX; j+=z2*i,z2=6-z2)
      isprime[j/24] &= imasks[(j/3)%8];
```

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

```
5   7   11  13  17  19  23  25  29  31  …
  +2  +4  +2  +4  +2  +4  +2  +4  +2   …
```

```
for (i=5,z1=2; i<=sqrt_value; i+=z1,z1=6-z1)
  if (isprime[i/24]&masks[(i/3)%8])
    for (j=i*i,z2=z1; j<=MAX; j+=z2*i,z2=6-z2)
      isprime[j/24] &= imasks[(j/3)%8];
```

- 常常應用程式需要質數表

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

```
5   7   11  13  17  19  23  25  29  31  …
  +2  +4  +2  +4  +2  +4  +2  +4  +2   …
```

```
for (i=5,z1=2; i<=sqrt_value; i+=z1,z1=6-z1)
  if (isprime[i/24]&masks[(i/3)%8])
    for (j=i*i,z2=z1; j<=MAX; j+=z2*i,z2=6-z2)
      isprime[j/24] &= imasks[(j/3)%8];
```

- 常常應用程式需要質數表

```
int primes[105097565], nprimes=2; primes[0]=2, primes[1]=3;
for (i=5,z1=2; i<=MAX; i+=z1,z1=6-z1)
  if ((isprime[i/24]&masks[(i/3)%8])!=0) primes[nprimes++] = i;
```

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

```
5   7   11  13  17  19  23  25  29  31  ...
  +2  +4  +2  +4  +2  +4  +2  +4  +2   ...
```

```
for (i=5,z1=2; i<=sqrt_value; i+=z1,z1=6-z1)
  if (isprime[i/24]&masks[(i/3)%8])
    for (j=i*i,z2=z1; j<=MAX; j+=z2*i,z2=6-z2)
      isprime[j/24] &= imasks[(j/3)%8];
```

- 常常應用程式需要質數表

```
int primes[105097565], nprimes=2; primes[0]=2, primes[1]=3;
for (i=5,z1=2; i<=MAX; i+=z1,z1=6-z1)
  if ((isprime[i/24]&masks[(i/3)%8])!=0) primes[nprimes++] = i;
```

- 4MB => 2MB, 小於 $2^{31}$ 的質數的最大差距是 **292**

# 程式實作 VII

- 6i+1 和 6i-1 可以合併成一個迴圈

```
5   7   11  13  17  19  23  25  29  31  …
  +2  +4  +2  +4  +2  +4  +2  +4  +2   …
```

```
for (i=5,z1=2; i<=sqrt_value; i+=z1,z1=6-z1)
  if (isprime[i/24]&masks[(i/3)%8])
    for (j=i*i,z2=z1; j<=MAX; j+=z2*i,z2=6-z2)
      isprime[j/24] &= imasks[(j/3)%8];
```

- 常常應用程式需要質數表

```
int primes[105097565], nprimes=2; primes[0]=2, primes[1]=3;
for (i=5,z1=2; i<=MAX; i+=z1,z1=6-z1)
  if ((isprime[i/24]&masks[(i/3)%8])!=0) primes[nprimes++] = i;
```

- 4MB => 2MB, 小於 $2^{31}$ 的質數的最大差距是 **292**

```
short primes[105097565]; int prev=3, nprimes=2; primes[0]=2, primes[1]=1;
for (i=5,z1=2; i<=MAX; i+=z1,z1=6-z1)
  if ((isprime[i/24]&masks[(i/3)%8])!=0) primes[nprimes++] = i-prev, prev=i;
```

# 分解因數

- 數字大的時候 ($2^{1024}$): 別試太久，這是個 NP 的問題，有百年的歷史，還是效率不好，目前運算力可以分解的數字大約在 $2^{512}$

# 分解因數

- 數字大的時候 ($2^{1024}$): 別試太久，這是個 NP 的問題，有百年的歷史，還是效率不好，目前運算力可以分解的數字大約在 $2^{512}$
- 數字不太大的時候 ($2^{64}$): 建質數表就可以分解它，程式競賽或是找工作面試題目基本上是這種

# 分解因數

- 數字大的時候 ($2^{1024}$): 別試太久，這是個 NP 的問題，有百年的歷史，還是效率不好，目前運算力可以分解的數字大約在 $2^{512}$
- 數字不太大的時候 ($2^{64}$): 建質數表就可以分解它，程式競賽或是找工作面試題目基本上是這種

```
int i, j, nfactors=0,factors[100], degrees[100];
for (i=0; primes[i]*primes[i]<=target; i++) {
   for (j=0; target%primes[i]==0; target/=primes[i]) j++;
   if (j>0) factors[nfactors] = primes[i], degrees[nfactors++] = j;
}
if (target>1) factors[nfactors] = target, degrees[nfactors++] = 1;
```

# 分解因數

- 數字大的時候 ($2^{1024}$): 別試太久，這是個 NP 的問題，有百年的歷史，還是效率不好，目前運算力可以分解的數字大約在 $2^{512}$
- 數字不太大的時候 ($2^{64}$): 建質數表就可以分解它，程式競賽或是找工作面試題目基本上是這種

```
int i, j, nfactors=0,factors[100], degrees[100];
for (i=0; primes[i]*primes[i]<=target; i++) {
   for (j=0; target%primes[i]==0; target/=primes[i]) j++;
   if (j>0) factors[nfactors] = primes[i], degrees[nfactors++] = j;
}
if (target>1) factors[nfactors] = target, degrees[nfactors++] = 1;
```

```
int i, j, nfactors=0,factors[100], degrees[100];
for (i=0,p=prime_diff[0]; p*p<=target; i++,p+=prime_diff[i]) {
   for (j=0; target%p==0; target/=p) j++;
   if (j>0) factors[nfactors] = p, degrees[nfactors++] = j;
}
if (target>1) factors[nfactors] = target, degrees[nfactors++] = 1;
```

# 如果數字不太大的話

由大到小輸出所有因數

# 如果數字不太大的話

由大到小輸出所有因數

直接用試除法

# 如果數字不太大的話

由大到小輸出所有因數

直接用試除法

```
#include <stdio.h>
int main() {
    int g, f;
    scanf("%d", &g);
    for (f=1; f*f<=g; f++)
        if (g% f==0)
            printf("d ", g/f);
    f--;
    if (f*f==g) f--;
    for (; f>=1; f--)
        if (g%f==0)
            printf("%d ", f);
    return 0;
}
```

# 如果數字不太大的話

由大到小輸出所有因數

直接用試除法

```
#include <stdio.h>
int main() {
    int g, f;
    scanf("%d", &g);
    for (f=1; f*f<=g; f++)
        if (g% f==0)
            printf("d ", g/f);
    f--;                                f*f>=g
    if (f*f==g) f--;        兩個迴圈
    for (; f>=1; f--)                   f*f<g
        if (g%f==0)
            printf("%d ", f);
    return 0;
}
```

# 如果數字不太大的話

由大到小輸出所有因數

直接用試除法

```
#include <stdio.h>
int main() {
    int g, f;
    scanf("%d", &g);
    for (f=1; f*f<=g; f++)
        if (g% f==0)
            printf("d ", g/f);
    f--;
    if (f*f==g) f--;
    for (; f>=1; f--)
        if (g%f==0)
            printf("%d ", f);
    return 0;
}
```

分解出所有因數

f*f>=g

兩個迴圈

f*f<g

# 如果數字不太大的話

## 由大到小輸出所有因數

直接用試除法

```
#include <stdio.h>
int main() {
    int g, f;
    scanf("%d", &g);
    for (f=1; f*f<=g; f++)
        if (g% f==0)
            printf("d ", g/f);
    f--;                          f*f>=g
    if (f*f==g) f--;      兩個迴圈
    for (; f>=1; f--)             f*f<g
        if (g%f==0)
            printf("%d ", f);
    return 0;
}
```

## 分解出所有因數

```
#include <stdio.h>
int main() {
    int i, f, g, nf=0,factors[100], degrees[100];
    scanf("%d", &g);
    for (f=2,nf=0; f*f<=g; f+=1+(f>2))  {
        for (i=0; g%f==0; i++) g/=f;
        if (i>0) factors[nf]=f, degrees[nf++]= i;
    }
    if (g>1) factors[nf] = g, degrees[nf++] = 1;
    for (i=0; i<nf; i++)
        printf("%d %d\n", factors[i], degrees[i]);
    return 0;
}
```

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

運用 {

1. 費瑪測試: 如果 $\exists a, a^{n-1} \neq 1 \pmod n$ 則 n 是合成數

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

運用
1. 費瑪測試: 如果 $\exists a, a^{n-1} \neq 1 \pmod n$ 則 n 是合成數
2. 分解因數基本定理: $\exists x,y, x^2 \equiv y^2 \pmod n$ 且 $x \neq \pm y \pmod n$，則 gcd(x+y) | n 或是 gcd(x-y) | n

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

  運用
  1. 費瑪測試: 如果 $\exists a, a^{n-1} \neq 1 \pmod n$ 則 n 是合成數
  2. 分解因數基本定理: $\exists x,y, x^2 \equiv y^2 \pmod n$ 且 $x \neq \pm y \pmod n$，則 $\gcd(x+y) \mid n$ 或是 $\gcd(x-y) \mid n$
- 方法

  1. 隨機挑選 a, 費瑪測試: $a^{n-1} \neq 1 \pmod n$, 則 n 不是質數

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

  運用
  1. 費瑪測試: 如果 $\exists a, a^{n-1} \neq 1 \pmod n$ 則 n 是合成數
  2. 分解因數基本定理: $\exists x,y, x^2 \equiv y^2 \pmod n$ 且 $x \neq \pm y \pmod n$，則 $\gcd(x+y) \mid n$ 或是 $\gcd(x-y) \mid n$

- 方法
  1. 隨機挑選 a, 費瑪測試: $a^{n-1} \neq 1 \pmod n$, 則 n 不是質數
  2. 如果 $a^{n-1} \equiv 1 \pmod n$, 將 n-1 表示為 $2^k m$, 令 $b_0 \equiv a^m \pmod n$

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

  運用
  1. 費瑪測試: 如果 $\exists a, a^{n-1} \neq 1 \pmod n$ 則 n 是合成數
  2. 分解因數基本定理: $\exists x,y, x^2 \equiv y^2 \pmod n$ 且 $x \neq \pm y \pmod n$，則 $\gcd(x+y) \mid n$ 或是 $\gcd(x-y) \mid n$

- 方法
  1. 隨機挑選 a, 費瑪測試: $a^{n-1} \neq 1 \pmod n$, 則 n 不是質數
  2. 如果 $a^{n-1} \equiv 1 \pmod n$, 將 n-1 表示為 $2^k m$, 令 $b_0 \equiv a^m \pmod n$
  3. 計算 $b_1, b_2, \ldots, b_i \equiv (b_{i-1})^2 \pmod n$ , 直到 $b_m \equiv a^{n-1} \pmod n$

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

  運用
  1. 費瑪測試: 如果 $\exists a, a^{n-1} \neq 1 \pmod n$ 則 n 是合成數
  2. 分解因數基本定理: $\exists x,y, x^2 \equiv y^2 \pmod n$ 且 $x \neq \pm y \pmod n$，則 $\gcd(x+y) \mid n$ 或是 $\gcd(x-y) \mid n$

- 方法

  1. 隨機挑選 a, 費瑪測試: $a^{n-1} \neq 1 \pmod n$, 則 n 不是質數
  2. 如果 $a^{n-1} \equiv 1 \pmod n$, 將 n-1 表示為 $2^k m$, 令 $b_0 \equiv a^m \pmod n$
  3. 計算 $b_1, b_2, \ldots, b_i \equiv (b_{i-1})^2 \pmod n$ , 直到 $b_m \equiv a^{n-1} \pmod n$

     過程中如果 $b_{i-1} \neq \pm 1 \pmod n$ 且 $b_i \equiv 1 \pmod n$ 則可應用分解因數的基本定理分解 n, 因此 n 不是質數

# Miller-Rabin 機率式測試

- 密碼學裡面用的數字都是很大的 ($2^{1024}$~$2^{8192}$)，怎麼知道一個數字時不是質數呢? 不能用暴力表列的 Eratosthenes 篩選法了
- 使用 Miller-Rabin 的機率式測試法

運用
1. 費瑪測試: 如果 $\exists a, a^{n-1} \neq 1 \pmod n$ 則 n 是合成數
2. 分解因數基本定理: $\exists x,y, x^2 \equiv y^2 \pmod n$ 且 $x \neq \pm y \pmod n$，則 $\gcd(x+y) \mid n$ 或是 $\gcd(x-y) \mid n$

- 方法

1. 隨機挑選 a, 費瑪測試: $a^{n-1} \neq 1 \pmod n$, 則 n 不是質數
2. 如果 $a^{n-1} \equiv 1 \pmod n$, 將 n-1 表示為 $2^k m$, 令 $b_0 \equiv a^m \pmod n$
3. 計算 $b_1, b_2, \ldots, b_i \equiv (b_{i-1})^2 \pmod n$ , 直到 $b_m \equiv a^{n-1} \pmod n$

   過程中如果 $b_{i-1} \neq \pm 1 \pmod n$ 且 $b_i \equiv 1 \pmod n$ 則可應用分解因數的基本定理分解 n, 因此 n 不是質數

重複上述 3 步驟 80 次，如果 n 不是質數，失敗的機率 < $2^{-80}$

# Miller-Rabin 實作

- 這個方法測試每一個數字需要花不少時間，尤其是當 n 是質數時，必須重複 80 次才能有信心它是質數

# Miller-Rabin 實作

- 這個方法測試每一個數字需要花不少時間，尤其是當 n 是質數時，必須重複 80 次才能有信心它是質數
- 基本上是為了沒有辦法暴力建表的大數字的測試方法，以 $2^{31}$ 範圍裡的質數來說，這樣子會比暴力建表還慢很多

# Miller-Rabin 實作

- 這個方法測試每一個數字需要花不少時間，尤其是當 n 是質數時，必須重複 80 次才能有信心它是質數
- 基本上是為了沒有辦法暴力建表的大數字的測試方法，以 $2^{31}$ 範圍裡的質數來說，這樣子會比暴力建表還慢很多
- 以 $2^{31}$ 範圍裡的質數來說

# Miller-Rabin 實作

- 這個方法測試每一個數字需要花不少時間，尤其是當 n 是質數時，必須重複 80 次才能有信心它是質數
- 基本上是為了沒有辦法暴力建表的大數字的測試方法，以 $2^{31}$ 範圍裡的質數來說，這樣子會比暴力建表還慢很多
- 以 $2^{31}$ 範圍裡的質數來說
  - 挑一個 a 測試過以後，漏網的合成數大概有 3000~5000 個

# Miller-Rabin 實作

- 這個方法測試每一個數字需要花不少時間，尤其是當 n 是質數時，必須重複 80 次才能有信心它是質數
- 基本上是為了沒有辦法暴力建表的大數字的測試方法，以 $2^{31}$ 範圍裡的質數來說，這樣子會比暴力建表還慢很多
- 以 $2^{31}$ 範圍裡的質數來說
  - 挑一個 a 測試過以後，漏網的合成數大概有 3000~5000 個
  - 挑兩個 a 測試，漏網的合成數大概有 400~1000 個，例如挑 73 和 277，會有 399 個合成數沒辦法確認，需要另外建一個表運用二分法來確認，要不然也可以賭賭看也許你的應用程式不會遇上剛好這 399 個裡面的質數 (共有 105,097,565 個質數, 0.00038% 的機會)

# Miller-Rabin 實作

- 這個方法測試每一個數字需要花不少時間，尤其是當 n 是質數時，必須重複 80 次才能有信心它是質數
- 基本上是為了沒有辦法暴力建表的大數字的測試方法，以 $2^{31}$ 範圍裡的質數來說，這樣子會比暴力建表還慢很多
- 以 $2^{31}$ 範圍裡的質數來說
  - 挑一個 a 測試過以後，漏網的合成數大概有 3000~5000 個
  - 挑兩個 a 測試，漏網的合成數大概有 400~1000 個，例如挑 73 和 277，會有 399 個合成數沒辦法確認，需要另外建一個表運用二分法來確認，要不然也可以賭賭看也許你的應用程式不會遇上剛好這 399 個裡面的質數 (共有 105,097,565 個質數, 0.00038% 的機會)
  - 挑三個 a (例如 {2,7,61}) 測試過以後，不是合成數的話就是質數，可是 $2^{31}$ 個數字中 6k±1 的每個數字都測試的話，比暴力建表慢很多

```
for (m=0,k=n-1; k%2==0; k/=2) m++;
```

```
for (m=0,k=n-1; k%2==0; k/=2) m++;

while (undecisive) {
   get a random a




}
```

```
for (m=0,k=n-1; k%2==0; k/=2) m++;

while (undecisive) {
   get a random a
   for (b=1,i=k,c=a; i>0; i/=2,c=(c*c)%n)
      if (i%2==1) b=(b*c)%n;




}
```

```
for (m=0,k=n-1; k%2==0; k/=2) m++;

while (undecisive) {
   get a random a
   for (b=1,i=k,c=a; i>0; i/=2,c=(c*c)%n)
      if (i%2==1) b=(b*c)%n;
   if (b==1||b==n-1) continue; // try another a
   for (i=0; i<m-1; i++) {
      b=(b*b)%n;
      if (b==1) // composite
         break;
      else if (b==x-1) // undecisive
         i=m;
   }
   if (i==m-1) {
      b=(b*b)%x;
      if (b!=1) break; // composite
   }
   if (i!=m+1) continue; // try another a
}
```