Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

MAT 685: C++ for Mathematicians

Prime pairs arrayed

John Perry

University of Southern Mississippi

Spring 2017

Outline

1. Prime pairs, yet again
2. Euler's totient function
   - A first factoring algorithm
   - A better factoring algorithm
   - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Outline

1. Prime pairs, yet again
2. Euler's totient function
   - A first factoring algorithm
   - A better factoring algorithm
   - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Classic problem

- Choose n
- Choose $a, b \in \{1, \ldots, n\}$
- Let p_n be probability that $\gcd(a, b) = 1$
- Does $\lim_{n\to\infty} p_n$ exist?
    - If so, what is its value?

Classic problem

- Choose n
- Choose $a, b \in \{1, \dots, n\}$
- Let p_n be probability that $\gcd(a, b) = 1$
- Does $\lim_{n\to\infty} p_n$ exist?
  - If so, what is its value?

Example

Let $n = 8$.

- Possible outcomes: $\binom{8}{2} = \frac{8!}{2!6!} = 28$
- Relatively prime pairs: $(1, 2)$, $(1, 3)$, ..., $(1, 8)$, $(2, 3)$, $(2, 5)$, $(2, 7)$, $(3, 4)$, $(3, 5)$, $(3, 7)$, $(3, 8)$, $(4, 5)$, $(4, 7)$, $(6, 7)$, $(7, 8)$
- So $p_8 = {}^{18}/_{28} = 9/14 \approx 64.3\%$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

A problem

Still too many numbers! Even the Monte Carlo algorithm takes too long at some point.

Outline

1 Prime pairs, yet again

2 Euler's totient function
- A first factoring algorithm
- A better factoring algorithm
- Back to the totient

3 Counting pairs

4 Answering the question

5 Summary

Get Euler's help!

Euler's "totient" function

$$\varphi(n) = |\{m \in \mathbb{N}\} : 1 \leq m \leq n \text{ and } \gcd(m,n) = 1| \ .$$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Get Euler's help!

Euler's "totient" function

$$\varphi(n) = |\{m \in \mathbb{N}\} : 1 \leq m \leq n \text{ and } \gcd(m, n) = 1| \ .$$

Example

n	2	3	4	5	6	7	8	9	10
$\varphi(n)$	1	2	2	4	2	6	4	6	4

n	11	12	13	14	15	16	17
$\varphi(n)$	10	4	12	6	8	8	16

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Notice anything?

n	2	3	4	5	6	7	8	9	10
$\varphi(n)$	1	2	2	4	2	6	4	6	4

n	11	12	13	14	15	16	17
$\varphi(n)$	10	4	12	6	8	8	16

- if $n = p$ is prime...
- if $n = p^k$ is a prime power...
- if $n = ab$ is a relatively prime product...

Notice anything?

n	2	3	4	5	6	7	8	9	10
$\varphi(n)$	1	2	2	4	2	6	4	6	4

n	11	12	13	14	15	16	17
$\varphi(n)$	10	4	12	6	8	8	16

- if $n = p$ is prime... $\varphi(p) = p - 1$
- if $n = p^k$ is a prime power...
- if $n = ab$ is a relatively prime product...

Notice anything?

n	2	3	4	5	6	7	8	9	10
$\varphi(n)$	1	2	2	4	2	6	4	6	4

n	11	12	13	14	15	16	17
$\varphi(n)$	10	4	12	6	8	8	16

- if $n = p$ is prime... $\varphi(p) = p - 1$
- if $n = p^k$ is a prime power... $\varphi(p^k) = p^k - p^{k-1}$
- if $n = ab$ is a relatively prime product...

Notice anything?

n	2	3	4	5	6	7	8	9	10
$\varphi(n)$	1	2	2	4	2	6	4	6	4

n	11	12	13	14	15	16	17
$\varphi(n)$	10	4	12	6	8	8	16

- if $n = p$ is prime... $\varphi(p) = p - 1$
- if $n = p^k$ is a prime power... $\varphi(p^k) = p^k - p^{k-1}$
- if $n = ab$ is a relatively prime product... $\varphi(ab) = \varphi(a)\varphi(b)$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

These properties make sense!

Theorem

If n is prime, then $\varphi(n) = n - 1$.

Proof.

Think about it a moment. □

Theorem

Proof.

□

These properties make sense!

Theorem

If n is prime, then $\varphi(n) = n - 1$.

Proof.

$1, 2, \ldots, n-1$ are all rel. prime to n. □

Theorem

Proof.

□

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

These properties make sense!

Theorem

If n is prime, then $\varphi(n) = n - 1$.

Proof.

$1, 2, \ldots, n - 1$ are all rel. prime to n. □

Theorem

If $n = p^k$ is a prime power, then $\varphi(n) = p^k - p^{k-1}$.

Proof.

Think about it a moment. □

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

These properties make sense!

Theorem

If n is prime, then $\varphi(n) = n - 1$.

Proof.

$1, 2, \ldots, n-1$ are all rel. prime to n. □

Theorem

If $n = p^k$ is a prime power, then $\varphi(n) = p^k - p^{k-1}$.

Proof.

Only $p, 2p, 3p, \ldots, (p^{k-1})\,p$ are not rel. prime to p. □

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

This one we can only sketch

Theorem

If $n = ab$ *is a relatively prime product, then* $\varphi(n) = \varphi(a)\varphi(b)$.

Proof.

Think about it a moment.

$\square$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

This one we can only sketch

Theorem

If $n = ab$ is a relatively prime product, then $\varphi(n) = \varphi(a)\varphi(b)$.

Proof.

Products of numbers not rel. prime to a or b are also not rel. prime to p. Numbers not rel. prime to p must also have a common factor with a or b (requires Chinese Remainder Theorem). □

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Non-trivial example

Compute $\varphi(100)$

Non-trivial example

Compute $\varphi(100)$

- $\varphi(100) = \varphi(2^2 \times 5^2)$

MAT 685: C++ for Mathematicians

John Perry

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Non-trivial example

Compute $\varphi(100)$

- $\varphi(100) = \varphi(2^2 \times 5^2)$
- $\varphi(100) = \varphi(2^2) \times \varphi(5^2)$

$$\varphi(ab) = \varphi(a)\,\varphi(b)$$

Non-trivial example

Compute $\varphi(100)$

- $\varphi(100) = \varphi\left(2^2 \times 5^2\right)$
- $\varphi(100) = \varphi\left(2^2\right) \times \varphi\left(5^2\right)$
- $\varphi(100) = \left(2^2 - 2^1\right) \times \left(5^2 - 5^1\right)$

$$\varphi(ab) = \varphi(a)\,\varphi(b)$$

$$\varphi\left(p^k\right) = p^k - p^{k-1}$$

Non-trivial example

Compute $\varphi(100)$

- $\varphi(100) = \varphi(2^2 \times 5^2)$
- $\varphi(100) = \varphi(2^2) \times \varphi(5^2)$ $\qquad \varphi(ab) = \varphi(a)\varphi(b)$
- $\varphi(100) = (2^2 - 2^1) \times (5^2 - 5^1)$ $\qquad \varphi(p^k) = p^k - p^{k-1}$
- $\varphi(100) = 40$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Non-trivial example

Compute $\varphi(100)$

- $\varphi(100) = \varphi(2^2 \times 5^2)$
- $\varphi(100) = \varphi(2^2) \times \varphi(5^2)$ $\qquad \varphi(ab) = \varphi(a)\varphi(b)$
- $\varphi(100) = (2^2 - 2^1) \times (5^2 - 5^1)$ $\qquad \varphi(p^k) = p^k - p^{k-1}$
- $\varphi(100) = 40$

Bingo!

The 40 numbers are

$$01, 03, 07, 09, \quad 11, 13, 17, 19, \quad \ldots, \quad 91, 93, 97, 99.$$

(Ten groups of 4.)

How does all this help?

$n = 10$: (1 extra for $(1,1)$)

$$
\begin{aligned}
p_n &= \frac{1 + 2 \times |\{\text{rel. prime pairs } (i,j), \; i < j\}|}{|(a,b) : 1 \leq a, b \leq 10|} \\
&= \frac{1 + 2 \times |\{(1,2)\} \cup \{(1,3), (2,3)\} \cup \cdots \cup \{(1,10), (3,10), \ldots, (9,10)\}|}{100} \\
&= \frac{1 + 2 \times (|\{(1,2)\}| + + \cdots + |\{(1,10), (3,10), \ldots, (9,10)\}|)}{100} \\
&= \frac{1 + 2 \times (\varphi(2) + \varphi(3) + \cdots + \varphi(10))}{100}
\end{aligned}
$$

How does all this help?

$n = 10$: (1 extra for $(1,1)$)

$$p_n = \frac{1 + 2 \times |\{\text{rel. prime pairs } (i,j), i < j\}|}{|(a,b) : 1 \le a, b \le 10|}$$
$$= \frac{1 + 2 \times |\{(1,2)\} \cup \{(1,3), (2,3)\} \cup \cdots \cup \{(1,10), (3,10), \ldots, (9,10)\}|}{100}$$
$$= \frac{1 + 2 \times (|\{(1,2)\}| + + \cdots + |\{(1,10), (3,10), \ldots, (9,10)\}|)}{100}$$
$$= \frac{1 + 2 \times (\varphi(2) + \varphi(3) + \cdots + \varphi(10))}{100}$$

In general,

$$p_n = \frac{1 + 2 \sum_{k=2}^{n} \varphi(k)}{n^2}$$

(book gives different, but equivalent, formula)

So what?

$$p_n = \frac{1 + 2\sum_{k=2}^{n} \varphi(k)}{n^2}$$

given n
let $s = 0$
for each $k \in \{2, \ldots, n\}$
  add $\varphi(k)$ to s
multiply 2 to s
add 1 to s
divide s by n^2
return s

So what?

$$p_n = \frac{1 + 2\sum_{k=2}^{n} \varphi(k)}{n^2}$$

given n
let $s = 0$
for each $k \in \{2, \ldots, n\}$
  add $\varphi(k)$ to s
multiply 2 to s
add 1 to s
divide s by n^2
return s

Example ($n = 10$)

start w/$s = 0$
$k = 2, 3, \ldots, 10$
  $s = 1, 3, 5, 9, 11, 17, 21, 27, 31$
multiply: $s = 62$
add: $s = 63$
divide: $s = 0.63$
return 0.63

Homework

p. 88 #5.1

Outline

1. Prime pairs, yet again
2. Euler's totient function
   - A first factoring algorithm
   - A better factoring algorithm
   - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Factoring n

To find $\varphi(n)$, we need n's factors

Question

How do we find them?

Factoring n

To find $\varphi(n)$, we need n's factors

Question

How do we find them?

given n
let L be a list of $\sqrt{n}$ zeroes
for each $i \in \{2, \ldots, \sqrt{n}\}$
  while $i \mid n$
    increment L_i
    replace n by n/i
return L

Example

$$n = 100$$
$$L = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$

Example

$n = 100$

$L = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

loop $i = 2$

Example

$n = 100$

$L = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)$

loop $i = 2$

$2 \mid 100$, so increment L_2 and replace n by 50

Example

$n = 100$

$L = (0, 2, 0, 0, 0, 0, 0, 0, 0, 0)$

loop $i = 2$

$2 \mid 100$, so increment L_2 and replace n by 50

$2 \mid 50$, so increment L_2 and replace n by 25

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Example

$n = 100$

$L = (0, 2, 0, 0, 0, 0, 0, 0, 0, 0)$

loop $i = 2$

- $2 \mid 100$, so increment L_2 and replace n by 50
- $2 \mid 50$, so increment L_2 and replace n by 25
- $2 \nmid 25$: **while** loop ends

Example

$n = 100$

$L = (0, 2, 0, 0, 0, 0, 0, 0, 0, 0)$

loop $i = 2$

- $2 \mid 100$, so increment L_2 and replace n by 50
- $2 \mid 50$, so increment L_2 and replace n by 25
- $2 \nmid 25$: **while** loop ends

loop $i = 3$

- $3 \nmid 25$: **while** loop ends

Example

$n = 100$

$L = (0, 2, 0, 0, 0, 0, 0, 0, 0, 0)$

loop $i = 2$

- $2 \mid 100$, so increment L_2 and replace n by 50
- $2 \mid 50$, so increment L_2 and replace n by 25
- $2 \nmid 25$: **while** loop ends

loop $i = 3$

- $3 \nmid 25$: **while** loop ends

loop $i = 4$

- $4 \nmid 25$: **while** loop ends

Example

$n = 100$

$L = (0, 2, 0, 0, 0, 0, 0, 0, 0, 0)$

loop $i = 2$

- $2 \mid 100$, so increment L_2 and replace n by 50
- $2 \mid 50$, so increment L_2 and replace n by 25
- $2 \nmid 25$: **while** loop ends

loop $i = 3$

- $3 \nmid 25$: **while** loop ends

loop $i = 4$

- $4 \nmid 25$: **while** loop ends

loop $i = 5$

Example

$n = 100$

$L = (0, 2, 0, 0, 1, 0, 0, 0, 0, 0)$

loop $i = 2$

- $2 \mid 100$, so increment L_2 and replace n by 50
- $2 \mid 50$, so increment L_2 and replace n by 25
- $2 \nmid 25$: **while** loop ends

loop $i = 3$

- $3 \nmid 25$: **while** loop ends

loop $i = 4$

- $4 \nmid 25$: **while** loop ends

loop $i = 5$

- $5 \mid 25$: increment L_5 and replace n by 5

Example

$n = 100$

$L = (0, 2, 0, 0, 2, 0, 0, 0, 0, 0)$

loop $i = 2$

- $2 \mid 100$, so increment L_2 and replace n by 50
- $2 \mid 50$, so increment L_2 and replace n by 25
- $2 \nmid 25$: **while** loop ends

loop $i = 3$

- $3 \nmid 25$: **while** loop ends

loop $i = 4$

- $4 \nmid 25$: **while** loop ends

loop $i = 5$

- $5 \mid 25$: increment L_5 and replace n by 5
- $5 \mid 5$: increment L_5 and replace n by 1

Example

$n = 100$

$L = (0, 2, 0, 0, 2, 0, 0, 0, 0, 0)$

loop $i = 2$

- $2 \mid 100$, so increment L_2 and replace n by 50
- $2 \mid 50$, so increment L_2 and replace n by 25
- $2 \nmid 25$: **while** loop ends

loop $i = 3$

- $3 \nmid 25$: **while** loop ends

loop $i = 4$

- $4 \nmid 25$: **while** loop ends

loop $i = 5$

- $5 \mid 25$: increment L_5 and replace n by 5
- $5 \mid 5$: increment L_5 and replace n by 1

...

return $(0, 2, 0, 0, 2, 0, 0, 0, 0, 0)$

What did we get?

$(0, 2, 0, 0, 2, 0, 0, 0, 0, 0)$ tells us

$$100 = 2^2 \times 5^2$$

From there, we can determine $\varphi(100)$ by passing through the loop.

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

OK, but... lists?

How do we track a *list* of numbers?

Use an **array**, a block of memory.

- need list of 25 `int`'s? `int A[25];`
- need n int's, but don't know n? `int A[n];`
  - compile w/ `-std=c++11`
- To initialize array to 0, declare instead `int A[n] { 0 };`

OK, but... lists?

How do we track a *list* of numbers?

Use an **array**, a block of memory.

- need list of 25 `int`'s? `int A[25];`
- need n int's, but don't know n? `int A[n];`
  - compile w/ `-std=c++11`
- To initialize array to 0, declare instead `int A[n] { 0 };`

Now ready to implement!

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Interface

Place in a new directory, `factoring`

Listing 1: `factoring.hpp`

```
#ifndef __FACTORING_HPP_
#define __FACTORING_HPP_

/**
    A basic factoring algorithm:
    iterate from 2 to sqrt(n).
    @param n the number to factor
    @param primes an array to contain the primes
    @warning initialize the elements of @c primes
        to zero!
*/
void factors(long n, long * primes);

#endif
```

Implementation

Place in same directory

Listing 2: `factoring.cpp`

```
#include "factoring.hpp"

void factors(long m, long * primes) {
    long n = m;
    for (long i = 2; n != 1 and i <= m/2; ++i)
    {
        while (n % i == 0) {
            primes[i] += 1;
            n /= i;
        }
    }
}
```

Test program

Place in same directory

Listing 3: `test_factoring.cpp` (p. 1)

```
#include <iostream>
using std::cin; using std::cout;
using std::endl;

#include "factoring.hpp"

int main() {
    long n;

    cout << "Enter a number to factor --> ";
    cin >> n;

    long m = n/2 + 1;
    long primes[m];
```

Test program

Place in same directory

Listing 4: `test_factoring.cpp` (p. 2)

```
    for (long i = 0; i < m; ++i)
      primes[i] = 0;

    factors(n, primes);

    for (long i = 0; i < m; ++i) {
        if (primes[i] != 0)
            cout << i << '^' << primes[i] << ' ';
    }
    cout << endl;
}
```

Compile, execute test

```
$ g++ -c factoring.cpp
$ g++ -o test_factoring -std=c++11 -lm \
    factoring.o test_factoring.cpp
$ ./test_factoring
Enter a number to factor --> 100
2^2 5^2
$ ./test_factoring
Enter a number to factor --> 10
2^1 5^1
```

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Things to watch out for

Big-time no-no

Don't forget `-std=c++11`

```
$ g++ -o test_factoring -lm factoring.o \
        test_factoring.cpp
test_factoring.cpp: In function 'int main()':
test_factoring.cpp:14:18: warning: extended initializer
lists only available with -std=c++11 or -std=gnu++11
    long primes[m] { 0 };
```

Things to watch out for

Big-time no-no

Don't forget `-std=c++11`

```
$ g++ -o test_factoring -lm factoring.o \
        test_factoring.cpp
test_factoring.cpp: In function 'int main()':
test_factoring.cpp:14:18: warning: extended initializer
lists only available with -std=c++11 or -std=gnu++11
    long primes[m] { 0 };
```

Big-time no-no

Don't forget `-lm`

(You may get away with that last one.)

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Homework

p. 89 #5.6, 5.7, 5.10

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Outline

1. Prime pairs, yet again
2. Euler's totient function
   - A first factoring algorithm
   - A better factoring algorithm
   - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

This algorithm is not especially efficient

Why not?

- finds n's prime factorization
- tests divisibility by non-primes
- we could do better if we start with primes

But...

How do we find primes?

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Sieve of Eratosthenes

given n
let L be a list of n booleans
    (initialize all L to *True*)
set L_1 to *False*
for each $i \in \{2, \ldots, \sqrt{n}\}$
    if L_i is *True*
        let $a = i$
        while $a < n$
            add i to a
            set L_a to *False*
return L

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Sieve of Eratosthenes

given n

let L be a list of n booleans

  (initialize all L to *True*)

set L_1 to *False*

for each $i \in \{2, \ldots, \sqrt{n}\}$

  if L_i is *True*

    let $a = i$

    while $a < n$

      add i to a

      set L_a to *False*

return L

Example

$n = 20$

beginning of loop

F			

Sieve of Eratosthenes

given n
let L be a list of n booleans
    (initialize all L to *True*)
set L_1 to *False*
for each $i \in \{2, \ldots, \sqrt{n}\}$
    if L_i is *True*
        let $a = i$
        while $a < n$
            add i to a
            set L_a to *False*
return L

Example

$n = 20$

$i = 2$ — mark out multiples of 2

F	2		F
	F		F
	F		F
	F		F
	F		F

Sieve of Eratosthenes

given n
let L be a list of n booleans
  (initialize all L to *True*)
set L_1 to *False*
for each $i \in \{2, \ldots, \sqrt{n}\}$
  if L_i is *True*
    let $a = i$
    while $a < n$
      add i to a
      set L_a to *False*
return L

Example

$n = 20$

$i = 3$ — mark out multiples of 3

F	2	3	F
	F		F
F	F		F
	F	F	F
	F		F

Sieve of Eratosthenes

given n
let L be a list of n booleans
    (initialize all L to *True*)
set L_1 to *False*
for each $i \in \{2, \ldots, \sqrt{n}\}$
    if L_i is *True*
        let $a = i$
        while $a < n$
            add i to a
            set L_a to *False*
return L

Example

$n = 20$

$i = 4$ — $L_4 = F$: not prime; ***skip!***

F	2	3	F
	F		F
F	F		F
	F	F	F
	F		F

Sieve of Eratosthenes

given n
let L be a list of n booleans
    (initialize all L to *True*)
set L_1 to *False*
for each $i \in \{2, \ldots, \sqrt{n}\}$
    if L_i is *True*
        let $a = i$
        while $a < n$
            add i to a
            set L_a to *False*
return L

Example

$n = 20$

$i = 5 > \sqrt{20}$ — end loop, true entries prime!

F	2	3	F
5	F	7	F
F	F	11	F
13	F	F	F
17	F	19	F

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Observation

Theorem

Factoring n requires us to test at most $\sqrt[4]{n}$ numbers for primality.

Proof.

- Prime factor of n must be smaller than $\sqrt{n}$.
- Sieve of Eratosthenes needs $\sqrt{m}$ tests to find all primes less than m.

$\therefore$ Need $\sqrt{\sqrt{n}} = \sqrt[4]{n}$ tests.

□

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Observation

Theorem

Factoring n requires us to test at most $\sqrt[4]{n}$ numbers for primality.

Proof.

- Prime factor of n must be smaller than $\sqrt{n}$.
- Sieve of Eratosthenes needs $\sqrt{m}$ tests to find all primes less than m.

$\therefore$ Need $\sqrt{\sqrt{n}} = \sqrt[4]{n}$ tests.

$\square$

Time to implement the sieve!

Creating lists on-the-fly

Two kinds of array allocation

static create array using `Type var[num];`

dynamic create array using
`Type * var = new Type[num];`

Creating lists on-the-fly

Two kinds of array allocation

static	create array using `Type var[num];`
dynamic	create array using `Type * var = new Type[num];`

Why two?

- static allocation...
  - *more efficient* if `num` is known constant (e.g., "25")
  - *unsafe* if data needed outside function
    - memory **will be** trashed!
- dynamic allocation
  - when finished w/array, requires `delete [] var;`
  - *safe* to pass outside function

Creating lists on-the-fly

Two kinds of array allocation

static create array using `Type var[num];`

dynamic create array using
`Type * var = new Type[num];`

Why two?

- static allocation...
  - *more efficient* if `num` is known constant (e.g., "25")
  - *unsafe* if data needed outside function
    - memory **will be** trashed!
- dynamic allocation
  - when finished w/array, requires `delete [] var;`
  - *safe* to pass outside function

sieve's list of primes: ***dynamic***

- don't know how many
- need to return to caller

Interface

Place in `factoring` directory

Listing 5: `sieve.hpp`

```
#ifndef __SIEVE_H_
#define __SIEVE_H_

/**
    Sieve of Eratosthenes:
    generate table of primes.
    @param n find primes <= n
    @param primes array of primes
    @return number of primes found */
long sieve(long n, long * & primes);

#endif
```

Implementation

In same directory

Listing 6: sieve.cpp (p. 1)

```
#include <cmath>
using std::sqrt;

#include "sieve.hpp"

long sieve(long n, long * & primes) {
    long m = long(sqrt(n));
    primes = new long[n];

    long num_primes = 0;
    bool * theSieve = new bool[n];
    for (long i = 2; i < n; ++i)
        theSieve[i] = true;
```

Implementation

In same directory

Listing 7: sieve.cpp (p. 2)

```
    long i;
    for (i = 2; i < m + 1; ++i) {
        if (theSieve[i] == true) {
            primes[num_primes] = i;
            ++num_primes;
            long a = i;
            while (a < n) {
                theSieve[a] = false;
                a += i;
            }
        }
    }
```

Implementation

In same directory

Listing 8: sieve.cpp (p. 3)

```
    for (/* */ ; i < n ; ++i) {
        if (theSieve[i] == true) {
            primes[num_primes] = i;
            ++num_primes;
        }
    }
    delete [] theSieve;
    return num_primes;
}
```

Test program

Place in same directory

Listing 9: test_sieve.cpp (p. 1)

```
#include <iostream>
using std::cin; using std::cout;
using std::endl;

#include "sieve.hpp"

int main() {
    long * primes;
    long n;

    cout << "This program finds all primes ";
    cout << "less than your choice of number.\n";
    cout << "Please choose a number --> ";
    cin >> n;
```

Test program

Place in same directory

Listing 10: `test_sieve.cpp` (p. 2)

```
    long np = sieve(n, primes);
    cout << "There are " << np << " primes:\n";
    for (long i = 0; i < np; ++i)
        cout << primes[i] << ", ";
    cout << endl;
    delete [] primes;
}
```

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Homework

p. 90 #5.11

Outline

1. Prime pairs, yet again
2. Euler's totient function
   - A first factoring algorithm
   - A better factoring algorithm
   - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Recall our algorithm for p_n

given n
let $s = 0$
for each $k \in \{2, \ldots, n\}$
  add $\varphi(k)$ to s
multiply 2 to s
add 1 to s
divide s by n^2
return s

...we still need to compute the totient

Interface

Place in `factoring` folder

Listing 11: `totient.hpp`

```
#ifndef __TOTIENT_HPP_
#define __TOTIENT_HPP_

long totient(long n, long * primes);

#endif
```

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

One more property

Theorem

$$\varphi(n) = n \times \left(\frac{p_1 - 1}{p_1}\right) \times \left(\frac{p_2 - 1}{p_2}\right) \times \cdots \times \left(\frac{p_\ell - 1}{p_\ell}\right)$$

where $p_1, p_2, \ldots, p_\ell$ are the prime factors of n.

Proof.

Think about it a moment...

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

One more property

Theorem

$$\varphi(n) = n \times \left(\frac{p_1 - 1}{p_1}\right) \times \left(\frac{p_2 - 1}{p_2}\right) \times \cdots \times \left(\frac{p_\ell - 1}{p_\ell}\right)$$

where $p_1, p_2, \ldots, p_\ell$ are the prime factors of n.

Proof.

Recall that

$$\varphi(n) = \left(p_1^{k_1} - p_1^{k_1 - 1}\right) \cdots \left(p_1^{k_1} - p_\ell^{k_\ell - 1}\right).$$

Factor each factor's common factor:

$$\varphi(n) = \left(p_1^{k_1 - 1} \cdots p_\ell^{k_\ell - 1}\right) \times \left[(p_1 - 1) \cdots (p_k - 1)\right].$$

The leftmost product can be rewritten as

$$\varphi(n) = \frac{n}{p_1 \cdots p_k} \times \left[(p_1 - 1) \cdots (p_k - 1)\right],$$

and we are done.

Implementation

Place in `factoring` folder

Listing 12: `totient.cpp`

```
#include <cmath>
using std::sqrt;

#include "totient.hpp"

long totient(long n, long * primes) {
    if (n < 0) return 0;

    long result = n;
    for (long i = 0; n != 1 and primes[i] <= n; ++i) {
        if (n % primes[i] == 0) {
            result /= primes[i];
            result *= primes[i] - 1;
        }
    }
    return result;
}
```

Test program

Place in `factoring` folder

Listing 13: test_totient.cpp

```
#include <iostream>
using std::cin; using std::cout;
using std::endl;

#include "sieve.hpp"
#include "totient.hpp"

int main() {
  long n;
  cout << "This programing computes ";
  cout << "the totient of an integer.\n";
  cout << "Please input a number --> ";
  cin >> n;
  long *primes;
  long np = sieve(n, primes);
  cout << totient(n, primes) << endl;
  delete [] primes;
}
```

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Compile, execute

```
$ g++ -c sieve.cpp totient.cpp
$ g++ -o test_totient sieve.o totient.o \
        test_totient.cpp
$ ./test_totient
Please input a number --> 100
40
```

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Outline

1. Prime pairs, yet again
2. Euler's totient function
   - A first factoring algorithm
   - A better factoring algorithm
   - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Recall our algorithm for p_n

given n
let $s = 0$
for each $k \in \{2, \ldots, n\}$
    add $\varphi(k)$ to s
multiply 2 to s
add 1 to s
divide s by n^2
return s

Implementation

Place in `relprime_pairs` folder

Listing 14: `totient_pairs.cpp` (p. 1)

```
#include <iostream>
using std::cin; using std::cout;
using std::endl;
#include <iomanip>
using std::setprecision;

#include "../factoring/sieve.hpp"
#include "../factoring/totient.hpp"

/**
    Calculates probability that two int's
    chosen in {1,2,...,n} are rel prime,
    up to n=10^6.
*/
```

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Implementation

Place in `relprime_pairs` folder

Listing 15: `totient_pairs.cpp` (p. 2)

```
int main() {
    const long N = 10000000;
    const long UPDATE = 100000;

    long * primes;
    long np = sieve(N, primes);

    long count = 0;
    cout << setprecision(20);
```

Implementation

Place in `relprime_pairs` folder

Listing 16: `totient_pairs.cpp` (p. 3)

```
    cout << setprecision(20);
    for (long k = 1; k <= N; ++k) {
        count += totient(k, primes);
        if (k % UPDATE == 0) {
            cout << k/1000 << " thousand \t";
            cout << double(2*count - 1)
                    / double(k*k) << endl;
        }
    }
    delete [] primes;
    return 0;
}
```

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Compiling, executing

```
$ g++ -Ofast -o totient_pairs \
    totient_pairs.cpp \
    ../factoring/sieve.o \
    ../factoring/totient.o
$ ./totient_pairs
100 thousand    0.60793015070000000488
200 thousand    0.6079299458750000044
300 thousand    0.60792774407777783185
...
800 thousand    0.60792796007343752329
900 thousand    0.60792736490740739708
1000 thousand   0.60792710478300004961
```

Outline

1. Prime pairs, yet again
2. Euler's totient function
   - A first factoring algorithm
   - A better factoring algorithm
   - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

The "point"

Question

So what *is* $\lim_{n\to\infty} p_n$?

We found

$$\lim_{n\to\infty} p_n \approx 0.607927.$$

Online Encyclopedia of Integer Sequences: $^6/_{\pi^2}$!

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

The "point"

Question

So what *is* $\lim_{n\to\infty} p_n$?

We found

$$\lim_{n\to\infty} p_n \approx 0.607927.$$

Online Encyclopedia of Integer Sequences: $6/\pi^2$!

???

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

How does π figure into this?

Basel problem (famous)

Find

$$\sum_{n=1}^{\infty} \frac{1}{n^2} .$$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

How does π figure into this?

Basel problem (famous)

Find

$$\sum_{n=1}^{\infty} \frac{1}{n^2} .$$

Solution: (Euler, 1734)

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} .$$

Proof.

Hard. Go bother Dr. Hornor or Dr. Ding or Dr. Kohl. $\square$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

That doesn't explain jack squat.

Look at

$$\sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\frac{1}{5^2}+\frac{1}{6^2}+\cdots$$

$$= \frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{2^4}+\frac{1}{5^2}+\frac{1}{2^2\times 3^2}+\cdots$$

$$= \prod_p \left(1+\frac{1}{p^2}+\frac{1}{p^4}+\frac{1}{p^6}+\cdots\right)$$

$$= \prod_p \left(\frac{1}{1-\frac{1}{p^2}}\right).$$

So

$$\therefore \prod_p \left(1-\frac{1}{p^2}\right) = \frac{1}{\sum 1/n^2} = \frac{6}{\pi^2}.$$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

...and?

Probability two integers:

- divisible by 2: $\frac{1}{2^2}$, so not: $1 - \frac{1}{2^2}$
- divisible by 3: $\frac{1}{3^2}$, so not: $1 - \frac{1}{3^2}$
- divisible by 5: $\frac{1}{5^2}$, so not: $1 - \frac{1}{5^2}$

...

so...

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

...and?

Probability two integers:

- divisible by 2: $\frac{1}{2^2}$, so not: $1 - \frac{1}{2^2}$
- divisible by 3: $\frac{1}{3^2}$, so not: $1 - \frac{1}{3^2}$
- divisible by 5: $\frac{1}{5^2}$, so not: $1 - \frac{1}{5^2}$

...

so...

Probability two large integers relatively prime:

$$\prod_{p<\text{larger}} \left(1 - \frac{1}{p^2}\right)$$

...and?

Probability two integers:

- divisible by 2: $\frac{1}{2^2}$, so not: $1 - \frac{1}{2^2}$
- divisible by 3: $\frac{1}{3^2}$, so not: $1 - \frac{1}{3^2}$
- divisible by 5: $\frac{1}{5^2}$, so not: $1 - \frac{1}{5^2}$

...

so...

Probability two large integers relatively prime:

$$\prod_{p<\text{larger}} \left(1 - \frac{1}{p^2}\right)$$

Thus

$$\lim_{n\to\infty} p_n = \prod_p \left(1 - \frac{1}{p^2}\right) = \frac{6}{\pi^2}.$$

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Homework

p. 90 #5.12

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Outline

1. Prime pairs, yet again
2. Euler's totient function
    - A first factoring algorithm
    - A better factoring algorithm
    - Back to the totient
3. Counting pairs
4. Answering the question
5. Summary

Prime pairs, yet again

Euler's totient function

A first factoring algorithm

A better factoring algorithm

Back to the totient

Counting pairs

Answering the question

Summary

Summary

- Math stuff
  - Euler totient function, properties
  - Sieve of Eratosthenes
  - Online Encyclopedia of Integer Sequences
  - π turns up in the strangest places!
- Programming stuff
  - arrays
  - static array creation
  - dynamic array creation
  - `new` and `delete []`