Voorbereiding Programmeerwedstrijden

najaar 2021

`https://liacs.leidenuniv.nl/~vlietrvan1/vbpw/`

Rudy van Vliet
kamer 140 Snellius, tel. 071-527 2876
rvvliet(at)liacs(dot)nl

college 3, 20 september 2021

Number Theory

Deadline huiswerkopgave 1:

donderdag 23 september, 09.15 strict

6.6.4. Expressions

3.8.3. Common Permutation

- understand the problem
- algorithm. . .
- boundary case. . .

7.1. Prime Numbers

- $p > 1$ only divisible by 1 and itself
- 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, . . .
- how many prime numbers?
- fundamental theorem of arithmetic: unique prime factorization
- prime vs. composite
- possible divisors. . .

7.1.1. Finding Primes

```
void prime_factorization (long long x)
{ long long i,     // candidate prime factor
             c;    // remaining product to factor

  c = x;
  while (c%2 == 0)
  { cout << ' ' << 2;
    c = c/2;
  }

  ...
```

7.1.1. Finding Primes

```
  i = 3;
  while (i <= sqrt(c)+1)
  { if (c%i == 0)
    { cout << ' ' << i;
      c = c/i;
    }
    else
      i += 2;
  }

  ...
}
```

7.1.1. Finding Primes

```
  i = 3;
  while (i <= sqrt(c)+1)
  { if (c%i == 0)
    { cout << ' ' << i;
      c = c/i;
    }
    else
      i += 2;
  }

  if (c>1)
    cout << ' ' << c;
  cout << endl;
}
```

Sieve of Eratosthenes

```
#include <vector>
#include <bitset>

const long long max_upperbound = 1000000000;
bitset<max_upperbound+1> bs;
vector<int> primes;
```

Sieve of Eratosthenes

```
// Create list of primes in [0..upperbound]
void sieve (long long upperbound)
{ long long i, j;

  bs.set ();            // set all bits to 1
  bs[0] = bs[1] = 0;    // except indices 0 and 1

  for (i=2;i<=upperbound;i++)
  { if (bs[i])
    { primes.push_back ((int)i);  // add i to vector containing list
       // cross out multiples of i starting from i*i
      for (j=i*i;j<=upperbound;j+=i)
        bs[j] = 0;
    }  // bs[i]
  }  // for i

}  // sieve
```

Given Factorization

- $3085500 = 2 * 2 * 3 * 5 * 5 * 5 * 11 * 11 * 17$
- how many divisors
- how many different orders of factorization
- constructing divisors / orders with backtracking

7.2.1. Greatest Common Divisor

- for simplifying fractions: $\frac{24}{36}$
- $\gcd(24, 36) = 12$
- Euclid's algorithm:
  - $\gcd(a, b) = \gcd(b, a \bmod b)$.
    Why?
  - $\gcd(a, 0) = a$ (if $a > 0$)

Euclid's Algorithm

$$\begin{aligned} \gcd(34398, 2132) &= \gcd(2132, 286) \\ &= \gcd(286, 130) \\ &= \gcd(130, 26) \\ &= \gcd(26, 0) = 26 \end{aligned}$$

Extended Euclidean Algorithm

$$a \cdot x + b \cdot y = gcd(a, b)$$

```
// Find gcd (a,b) and x and y such that a*x + b*y = gcd (a,b)
int gcd (int a, int b, int &x, int &y)
{ int x1, y1;    // previous coefficients
  int g;         // value of gcd (a, b)

  if (b > a)
    return gcd (b, a, ..., ...);

  if (b == 0)
  { x = ...;
    y = ...;
    return a;
  }

  g = gcd (b, a%b, x1, y1);
  ...
  ...
  return g;
}
```

```
// Find gcd (a,b) and x and y such that a*x + b*y = gcd (a,b)
int gcd (int a, int b, int &x, int &y)
{ int x1, y1;    // previous coefficients
  int g;         // value of gcd (a, b)

  if (b > a)
    return gcd (b, a, y, x);

  if (b == 0)
  { x = 1;
    y = 0;
    return a;
  }

  g = gcd (b, a%b, x1, y1);
  x = y1;
  y = x1 - (a/b)*y1;
  return g;
}
```

```
// Find gcd (a,b) and x and y such that a*x + b*y = gcd (a,b)
int gcd (int a, int b, int &x, int &y)
{ int x1, y1;    // previous coefficients
  int g;         // value of gcd (a, b)

  if (b > a)
    return gcd (b, a, y, x);

  if (b == 0)
  { x = 1;
    y = 1000;
    return a;
  }

  g = gcd (b, a%b, x1, y1);
  x = y1;
  y = x1 - (a/b)*y1;
  return g;
}
```

7.6.3. Euclid Problem

7.6.3. Euclid Problem

- find *a* solution of $AX + BY = D$
- either $X > 0$ and $Y \leq 0$,
  or $X \leq 0$ and $Y > 0$
- 'next' solution is $A(X + \frac{B}{D}) + B(Y - \frac{A}{D}) = D$
- if $X > Y$, then decrease X and increase Y

$$\left\lceil \frac{X - Y}{\frac{A+B}{D}} \right\rceil \quad \text{times}$$

- if $X < Y$, then increase X and decrease Y

$$\left\lfloor \frac{Y - X}{\frac{A+B}{D}} \right\rfloor \quad \text{times}$$

7.2.2. Least Common Multiple

- for simultaneous periodicity of two distinct periodic events
- $\text{lcm}(24, 40) = 120$
- in general, $\text{lcm}(a, b) = \ldots$

7.2.2. Least Common Multiple

- for simultaneous periodicity of two distinct periodic events
- $\text{lcm}(24, 40) = 120$
- in general, $\text{lcm}(a, b) = \frac{ab}{\text{gcd}(a,b)} = a\frac{b}{\text{gcd}(a,b)}$

High-Precision Integers

- `__int128_t n;`

  if 128 bits is sufficient

- ```
  include <boost/multiprecision/cpp_int.hpp>
  using boost::multiprecision::cpp_int;

  cpp_int n;
  ```

- array of digits

- linked list of digits
  ```

7.3. Modular Arithmetic

- sometimes remainder modulo a number is sufficient (also in programming contest)

- $(x + y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$

$$(12345 + 9467) \bmod 100 \;=\; \ldots$$

7.3. Modular Arithmetic

- sometimes remainder modulo a number is sufficient
  (also in programming contest)

- $(x + y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$

  $$\begin{aligned}(12345 + 9467) \bmod 100 \\ &= ((12345 \bmod 100) + (9467 \bmod 100)) \bmod 100 \\ &= (45 + 67) \bmod 100 \\ &= 12\end{aligned}$$

- $(x - y) \bmod n = ((x \bmod n) - (y \bmod n)) \bmod n$

- $(x * y) \bmod n = ((x \bmod n) * (y \bmod n)) \bmod n$

- division: more complicated

7.3. Modular Arithmetic

Some applications:

- Finding the last digit: $2^{100} \bmod 10 = \ldots$
- RSA Encryption Algorithm: $m^k \bmod n$ with huge integers

7.6.2. Carmichael Numbers

7.6.2. Carmichael Numbers

- is n is prime, . . .
- if n is non-prime
  - for a is 2 to $n-1$ (as long as . . .)
    - compute a^n mod n
- $n < 65000$
  - long long
  - efficient exponentiation