

Vorbereitung Programmierwedstrijden

najaar 2023

<https://liacs.leidenuniv.nl/~vlietrvan1/vbpw/>

Rudy van Vliet

kamer 140 Snellius, tel. 071-527 2876

rvvliet(at)liacs(dot)nl

college 3, 19 september 2023

Number Theory

Deadline huiswerkopgave 1:

vrijdag 22 september, 23.59 **strict, en in praktijk misschien eerder. . .**

6.1. Basic Counting Techniques

- product rule: $|A| \times |B|$
5 shirts and 4 pants
- sum rule: $|A| + |B|$
5 shirts and 4 pants
- with overlap

$$|A \cup B| = |A| + |B| \dots$$

6.1. Basic Counting Techniques

- product rule: $|A| \times |B|$
5 shirts and 4 pants

- sum rule: $|A| + |B|$
5 shirts and 4 pants

- with overlap, inclusion and exclusion

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

6.6.2. How Many Pieces of Land

6.6.2. How Many Pieces of Land

-

$$a_n = a_{n-1} + \sum_{i=0}^{n-2} (1 + i * (n - 2 - i))$$

- $a_0 = 1$
- 1, 1, 2, 4, 8, 16, 31, 57, 99, 163, 256, 386, ...
- time complexity of algorithm...
- $0 \leq n \leq 2^{31}$
- OEIS:

$$a_n = (n^4 - 6n^3 + 23n^2 - 18n + 24)/24$$

with 128 bit integers

Common Combinatorial Objects

- permutations: $n!$
 $10! = 3,628,800$
- subsets: 2^n
 $2^{20} = 1,048,576$
- strings (sequences, repetition allowed): m^n

exhaustive search...

6.3. Binomial Coefficients

$\binom{n}{k}$, for

- k -member committees from n people
- paths across an $n \times m$ grid
- coefficients of $(a + b)^n$

- Pascal's triangle

					1					
				1		1				
			1		2		1			
		1		3		3		1		
	1		4		6		4		1	
1		5		10		10		5		1

Computing Binomial Coefficient

-

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

overflow

-

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

```

long long binomial_coefficient (int n, int m) // compute n choose m
{ int i, j;
  long long bc[MAXN+1][MAXN+1]; // table of binomial coefficients

  for (i=0;i<=n;i++)
    bc[i][0] = 1;

  for (i=0;i<=n;i++)
    bc[i][i] = 1;

  for (i=2;i<=n;i++)
    for (j=1;j<i;j++)
      bc[i][j] = bc[i-1][j-1] + bc[i-1][j];

  return bc[n][m];
}

```

pre: $0 \leq m \leq n \leq \text{MAXN}$
dynamic programming!

6.4. Other Counting Sequences

- Fibonacci numbers F_n

- $F_n = F_{n-1} + F_{n-2}$

- $F_0 = 0, F_1 = 1$

- $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots$

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right)$$

6.4. Other Counting Sequences

- Catalan numbers C_n
 - balanced formula of n pairs of brackets
 - $n = 3$: $((()))$, $(()())$, $(())()$, $()(())$, $()()()$
 - recurrence relation...

6.4. Other Counting Sequences

- Catalan numbers C_n
 - balanced formula of n pairs of brackets
 - $((()))$, $(())()$, $(())()$, $()(())$, $()()()$
 -

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

- $C_0 = 1$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, ...
- Online Encyclopedia of Integer Sequences

6.4. Other Counting Sequences

- Catalan numbers C_n
 - balanced formula of n pairs of brackets
 - $((()))$, $(())()$, $(())()$, $()(())$, $()()()$
 -

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

- $C_0 = 1$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, ...

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

6.6.4. Expressions

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: $\frac{24}{36}$
- $\text{gcd}(24, 36) = 12$
- Euclid's algorithm:
 - $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$.
Why?
 - $\text{gcd}(a, 0) = a$ (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  $a*x + b*y = \text{gcd}(a,b)$ 
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

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

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

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

7.2.2. Least Common Multiple

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

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)
- $(x + y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$
 $(12345 + 9467) \bmod 100 = \dots$

7.3. Modular Arithmetic

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

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

$$(12345 + 9467) \bmod 100$$

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

$$= (45 + 67) \bmod 100$$

$$= 12$$

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

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

- division: more complicated

7.3. Modular Arithmetic

Some applications:

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

7.6.2. Carmichael Numbers

7.6.2. Carmichael Numbers

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