Discrete Algebraic Structures

WiSe 2025/2026

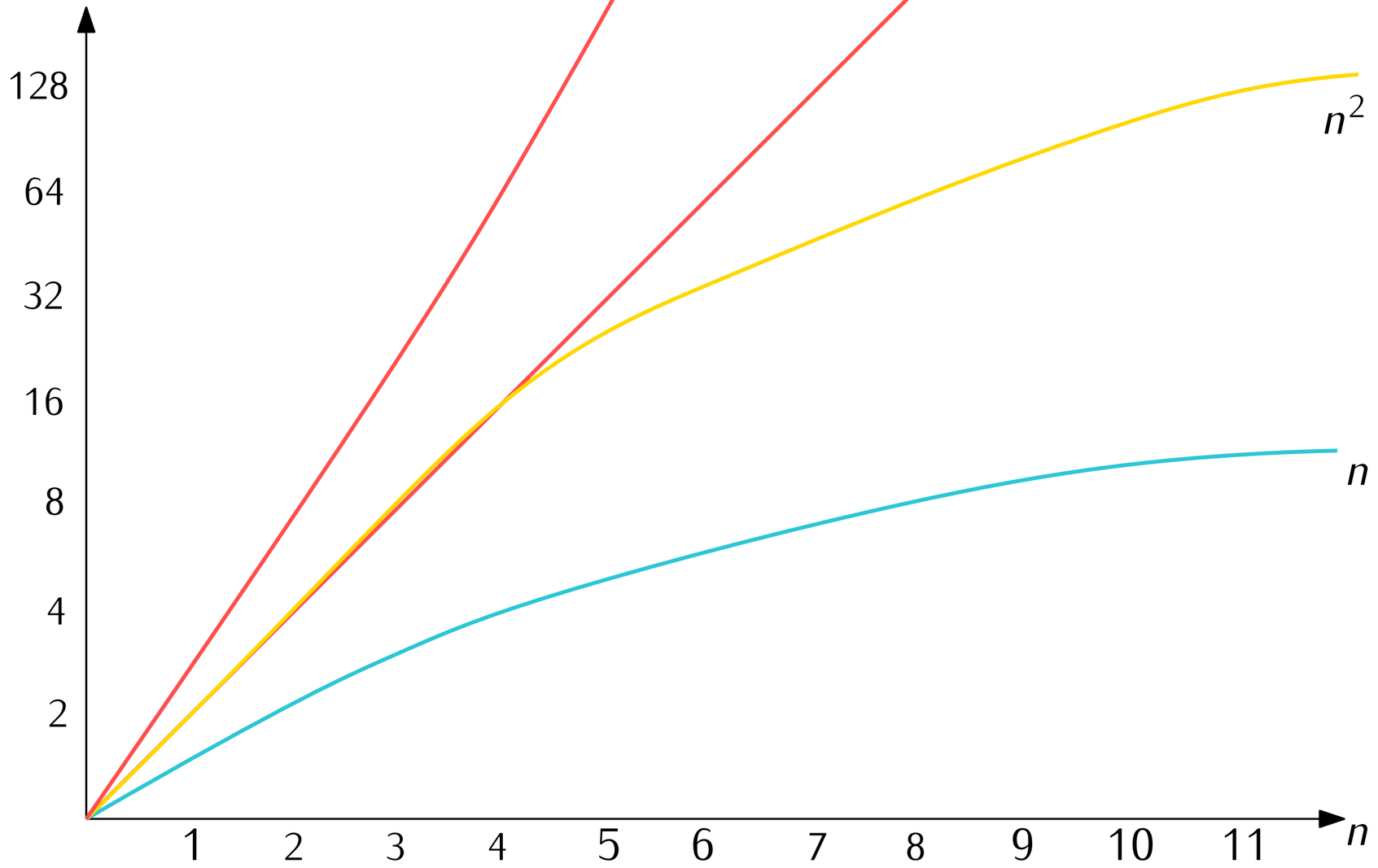
Prof. Dr. Antoine Wiehe
Research Group for Theoretical Computer Science

Theorem. The number of partitions of $\{1, \ldots, n\}$ into k parts is

$$\frac{1}{k!} \sum_{r=0}^{k} (-1)^r \binom{k}{r} (k-r)^n$$

Any idea how big this is?

How about k^n?



Theorem. The number of partitions of $\{1, \ldots, n\}$ into k parts is

$$\frac{1}{k!} \sum_{r=0}^{k} (-1)^r \binom{k}{r} (k-r)^n$$

Any idea how big this is?

How about k^n?

Definition. Let $f, g : \mathbb{N} \to \mathbb{N}$. We say $f \in O(g)$ if there are $C, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) \leq C \cdot g(n)$.

Example. $4n^2 + n + 1 \in O(n^2)$

Theorem. The number of partitions of $\{1, \dots, n\}$ into k parts is

$$\frac{1}{k!} \sum_{r=0}^{k} (-1)^r \binom{k}{r} (k-r)^n$$

Any idea how big this is?

How about k^n?

Definition. Let $f, g : \mathbb{N} \to \mathbb{N}$. We say $f \in O(g)$ if there are $C, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) \leq C \cdot g(n)$.

Exercise. Prove that the relation $\{(f, g) \mid f \in O(g)\}$ is a quasiorder.

	Order matters	Order does not matter
Replacement	n^k	$\binom{n+k-1}{n-1} \in O(n^k)$
No replacement	$n^{\underline{k}} \in O(n^k)$	$\frac{n!}{k!(n-k)!} \in O(n^k)$

Take away message:
$O(n^k)$ is "ok" for algorithms if k small
Don't ever try $O(k^n)$, even if k small

Number of surjective functions:
(from a set of size n to a set of size k)

$$O\left(\sum_{r=0}^{k} (-1)^r \binom{k}{r} (k-r)^n \right) \ni k^n$$

Number of partitions:
(of a set of size n into k parts)

$$O\left(\frac{1}{k!} \sum_{r=0}^{k} (-1)^r \binom{k}{r} (k-r)^n \right) \ni k^n$$

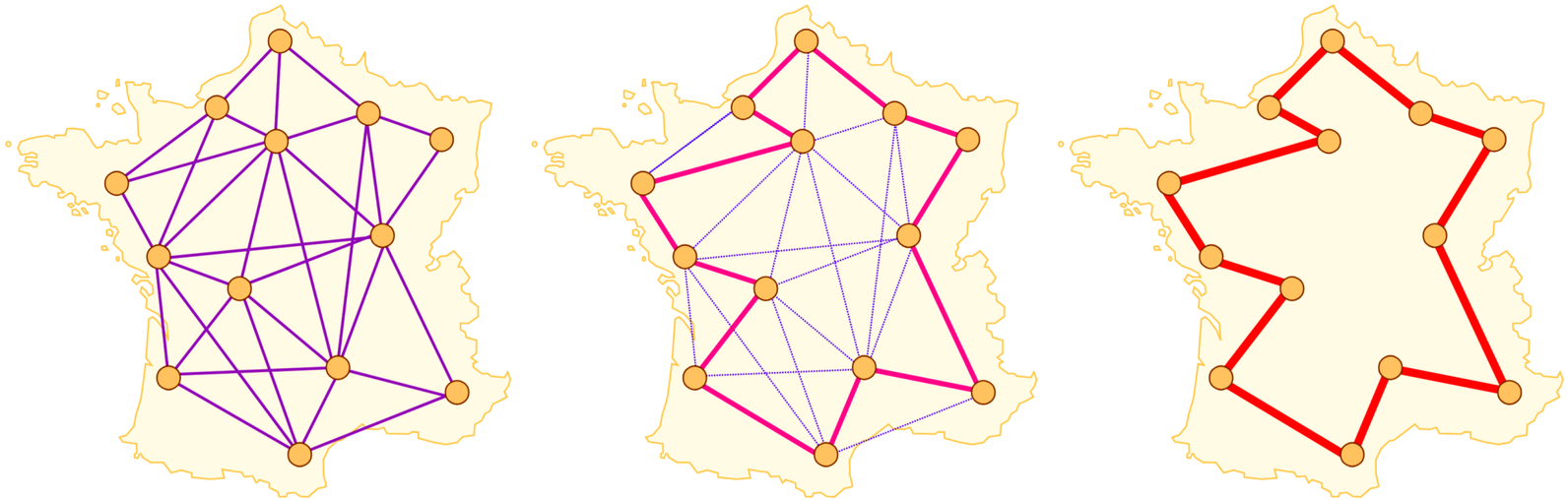
$f(n)$: runtime (say, in seconds) of a program when the input has size n

Efficient algorithm: when $f \in O(n^k)$ for $k \in \mathbb{N}$

- $f \in O(\log(n))$: locate an item in a sorted array
- $f \in O(n)$: find shortest path in a directed graph, find smallest element/biggest element/median in an array
- $f \in O(n \log(n))$: sort an array
- $f \in O(n^3)$: compute the multiplication of two matrices of size n

Problems where **no efficient algorithm** is known:

- $O(2^n)$: problem of finding a satisfying assignment for a propositional formula, optimal clustering of a set of n points into two clusters
- $O(n!)$: finding optimal tour visiting cities



- Countable sets, uncountable sets
- Countable sets = what can be represented exactly on a computer
- Combinatorial proofs as a way to prove equalities/inequalities about numbers using functions

$$\begin{aligned} \text{Injection } A \to B &\quad \leftrightarrow \quad |A| \leq |B| \\ \text{Surjection } A \to B &\quad \leftrightarrow \quad |A| \geq |B| \\ \text{Bijection } A \to B &\quad \leftrightarrow \quad |A| = |B| \end{aligned}$$

- Drawing a tuple/(multi)set with/without replacement
  n = size of the set we are drawing from
  k = number of draws
- How to use double counting
- The inclusion-exclusion principle for 2 and 3 sets
- How to apply the pigeonhole principle

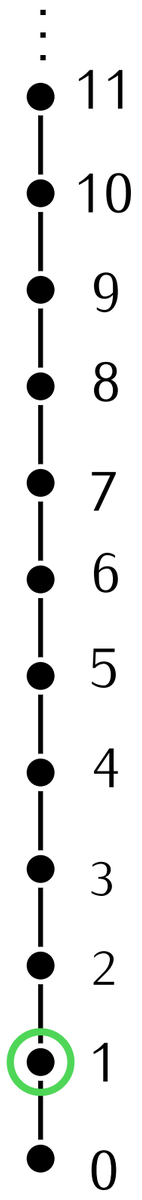
	Order matters	Order does not matter
Replacement	n^k	$\binom{n+k-1}{n-1}$
No replacement	$n^{\underline{k}}$	$\frac{n!}{k!(n-k)!}$

Elementary Number Theory

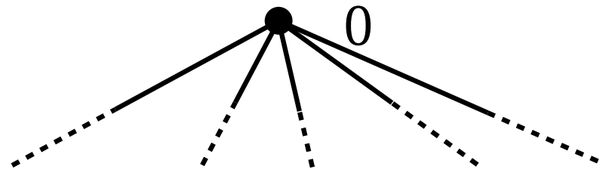
$$R = \{(n, m) \in \mathbb{N}_0^2 \mid n \leq m\}$$

- Linear
- Minimal elements: 0
- Maximal elements: None
- $n \wedge m$ always exists: $\min(n, m)$
- $n \vee m$ always exists: $\max(n, m)$

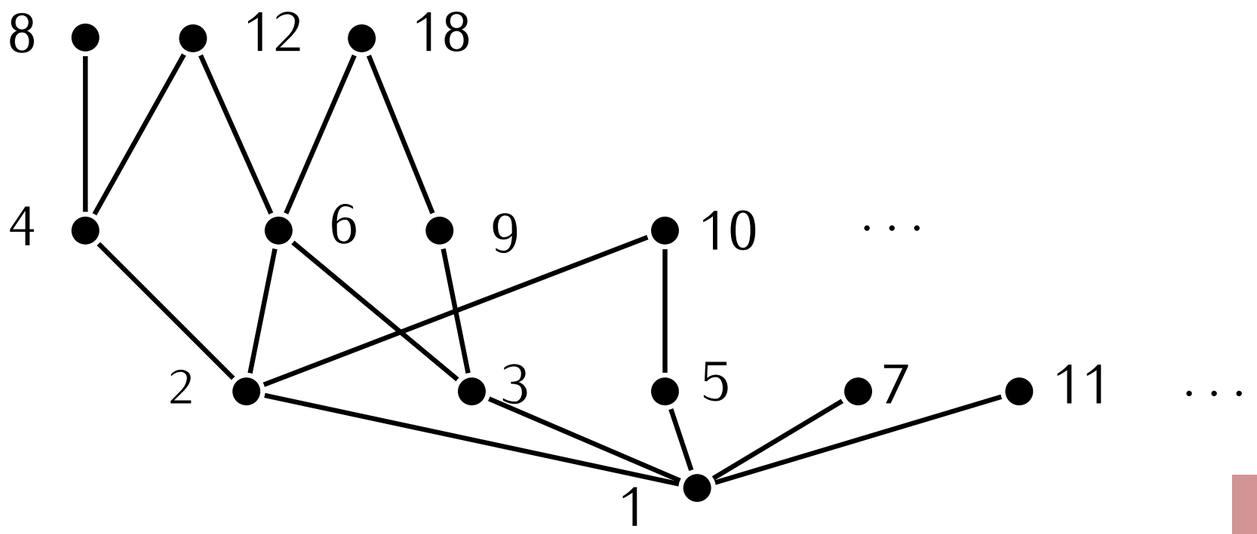
Every number n can be written (uniquely!) as $1 + 1 + 1 + \ldots$



$$R = \{(n, m) \in \mathbb{N}_0^2 \mid n \text{ divides } m\}$$

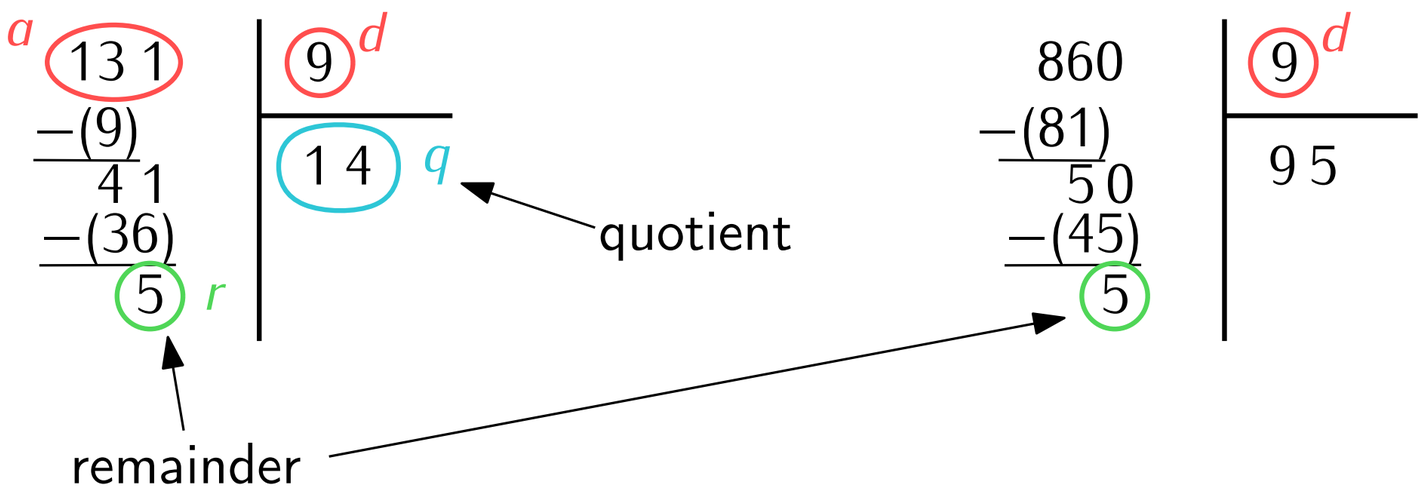


- Linear?
- Minimal elements?
- Maximal elements?
- $n \wedge m$?
- $n \vee m$?



Theorem. For all $a, d \in \mathbb{Z}$ such that $a \neq 0$, there exists a unique pair $(q, r) \in \mathbb{Z}^2$ such that:

- $a = q \cdot d + r$
- $r \in \{0, \ldots, |d| - 1\}$



Almost the same as `divmod` in Python:

```
divmod(131,9) # (14,5)
divmod(10,-3) # (-4,-2)
```

- If remainder is 0: we say d divides a
  a is a multiple of d

Notation: $d \mid a$

- Define $b \equiv_d b'$ by "they have the same remainder in the division by d"
  $131 \equiv_9 860$

What does 131 really *mean*? $132 = 1 \times 100 + 3 \times 10 + 2 \times 1 = \sum_{i=0}^{2} n_i \times 10^i$

Theorem. Let $b \in \mathbb{N}$ be such that $b \geq 2$. For every $n \in \mathbb{N}_0$, there exists a unique sequence $n_0, \ldots, n_k$ of numbers in $\{0, \ldots, b-1\}$ such that $n = \sum_{i=0}^{k} n_i b^i$.

This is the decomposition of n in base b, written $(n_k, \ldots, n_1, n_0)_b$.

- Computers use bits 0 and 1 $\rightsquigarrow$ base 2
  $(131)_{10} = (10000011)_2$

What is the largest number that can be represented with 8 bits?
(So binary decomposition has length at most 8.)

What does 131 really *mean*? $132 = 1 \times 100 + 3 \times 10 + 2 \times 1 = \sum_{i=0}^{2} n_i \times 10^i$

Theorem. Let $b \in \mathbb{N}$ be such that $b \geq 2$. For every $n \in \mathbb{N}_0$, there exists a unique sequence $n_0, \ldots, n_k$ of numbers in $\{0, \ldots, b-1\}$ such that $n = \sum_{i=0}^{k} n_i b^i$.

This is the decomposition of n in base b, written $(n_k, \ldots, n_1, n_0)_b$.

- Computers use bits 0 and 1 $\rightsquigarrow$ base 2
  $(131)_{10} = (10000011)_2$
- Base 16 also very common: `commit 6d068c500bd1baede277a8c1f62d1a7b5d1a1d12`
  $= 622426021449319981362286421400854052354972589330$

  about 17% shorter to write numbers in base 16 instead of 10, 75% shorter than in base 2
  alphabet $0, \ldots, 9, a, b, c, d, e, f$

$$(bad)_{16} = b \times 16^2 + a \times 16^1 + d \times 16$$
$$= 11 \times 256 + 10 \times 16 + 13 = 2816 + 160 + 13 = 2989$$

- $\log_b(n)$: number in $[k, k+1)$ when $b^k \leq n < b^{k+1}$
  $3 < log_{16}(2989) < 4$

Definition. Let $\leq$ be an order on A, $S \subseteq A$, and $a \in A$. We say:

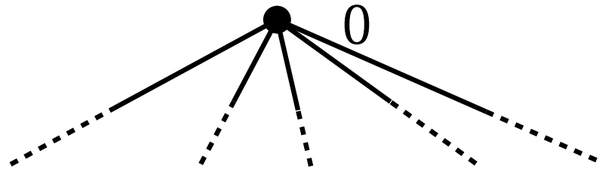
- a is a lower bound of S if for all $s \in S$, we have $a \leq s$
- a is a greatest lower bound of S if it is a lower bound and for every lower bound b of S, we have $b \leq a$.

We write $a \wedge b$ for the greatest lower bound of $\{a, b\}$, if it exists.

Definition. Let $S \subseteq \mathbb{N}_0$, and $d \in \mathbb{N}_0$. We say:

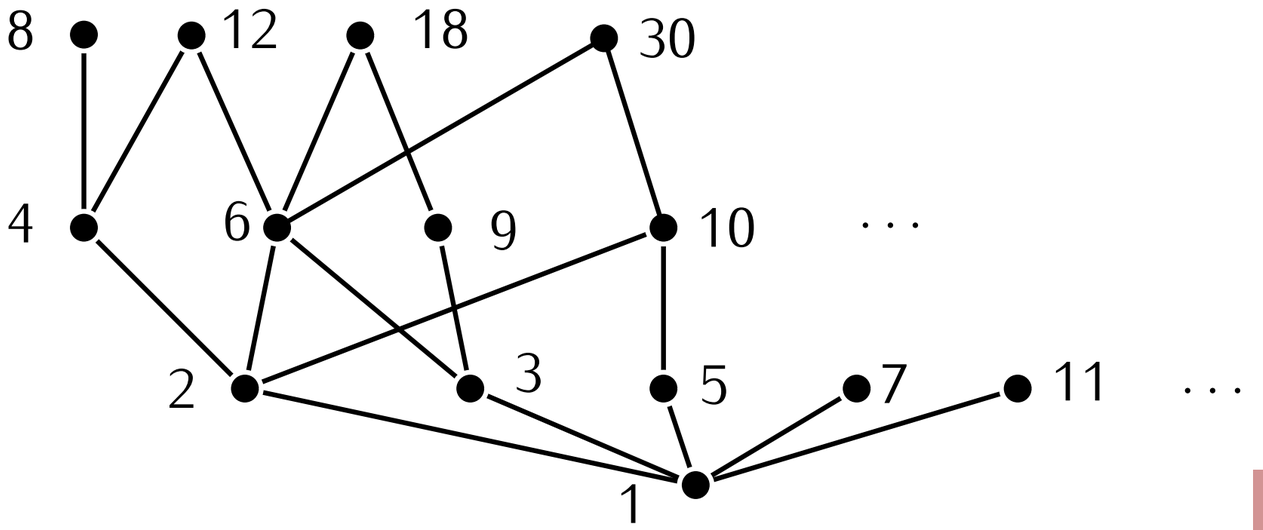
- d is a common divisor of S if for all $s \in S$, we have d divides s
- d is a greatest common divisor of S if it is a common divisor and for every common divisor d' of S, we have d' divides d

We write $a \wedge b$ for the greatest common divisor (gcd) of $\{a, b\}$.



What is $6 \wedge 10$?

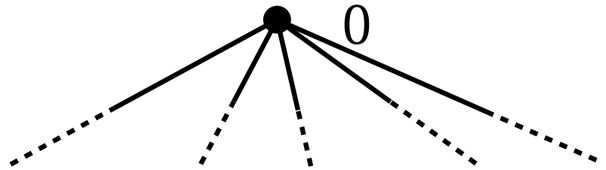
- 2
- 4
- 6
- 10
- 16



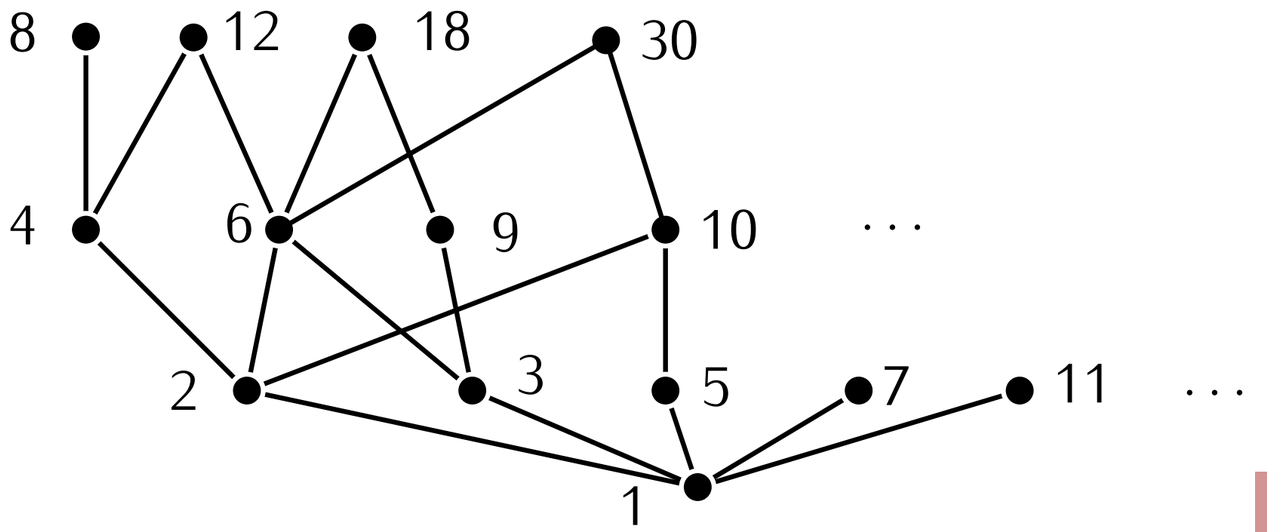
Definition. Let $S \subseteq \mathbb{N}_0$, and $d \in \mathbb{N}_0$. We say:

- m is a common multiple of S if for all $s \in S$, we have m is a multiple of s
- m is a lowest common multiple of S if it is a common multiple and for every common multiple m' of S, we have m divides m'

We write $a \vee b$ for the lowest common multiple (lcm) of $\{a, b\}$.



What is $4 \vee 10$?
(might not be on the picture!)



Input: two numbers $a, b \in \mathbb{N}_0$

Output: $a \wedge b$ (or $\gcd(a, b)$) ⟶ In particular, $\gcd(a, b)$ **always** exists!

Euclid
(probably lived around -300)

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Input: two numbers $a, b \in \mathbb{N}_0$

Output: $a \wedge b$ (or $\gcd(a, b)$) ⟶ In particular, $\gcd(a, b)$ **always** exists!

On input $a = 7, b = 24$:

$$24 = 3 \times 7 + 3$$

`remainders` $= (24, 7)$

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Input: two numbers $a, b \in \mathbb{N}_0$

Output: $a \wedge b$ (or $\gcd(a, b)$) ⟶ In particular, $\gcd(a, b)$ **always** exists!

On input $a = 7, b = 24$:

$$24 = 3 \times 7 + 3$$
$$7 = 2 \times 3 + 1$$

$\texttt{remainders} = (24, 7, 3)$

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Input: two numbers $a, b \in \mathbb{N}_0$

Output: $a \wedge b$ (or $\gcd(a, b)$) ⟶ In particular, $\gcd(a, b)$ **always** exists!

On input $a = 7, b = 24$:

$$24 = 3 \times 7 + 3$$
$$7 = 2 \times 3 + 1$$
$$3 = 3 \times 1 + 0$$
$$\texttt{remainders} = (24, 7, 3, 1)$$

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Input: two numbers $a, b \in \mathbb{N}_0$

Output: $a \wedge b$ (or $\gcd(a, b)$) ⟶ In particular, $\gcd(a, b)$ **always** exists!

On input $a = 7, b = 24$:

$$24 = 3 \times 7 + 3$$
$$7 = 2 \times 3 + 1$$
$$3 = 3 \times 1 + 0$$

$\texttt{remainders} = (24, 7, 3, 1, 0)$

this is $\gcd(7, 24)$

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

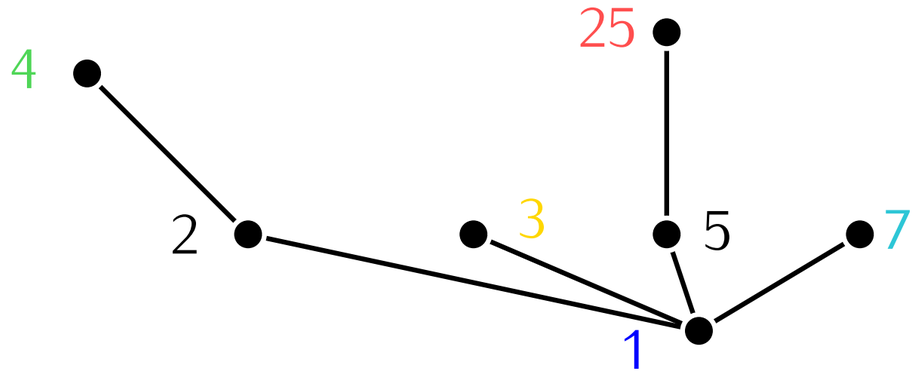
Input: two numbers $a, b \in \mathbb{N}_0$

Output: $a \wedge b$ (or $\gcd(a, b)$) ⟶ In particular, $\gcd(a, b)$ **always** exists!

On input $a = 7, b = 25$:

$$25 = 3 \times 7 + 4$$
$$7 = 1 \times 4 + 3$$
$$4 = 1 \times 3 + 1$$
$$3 = 3 \times 1 + 0$$

$\texttt{remainders} = (25, 7, 4, 3, 1, 0)$




```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Input: two numbers $a, b \in \mathbb{N}_0$
Output: $a \wedge b$ (or $\gcd(a,b)$) ⟶ In particular, $\gcd(a,b)$ **always** exists!

On input $a = 7, b = 25$:

$$25 = 3 \times 7 + 4$$
$$7 = 1 \times 4 + 3$$
$$4 = 1 \times 3 + 1$$
$$3 = 3 \times 1 + 0$$
$$\texttt{remainders} = (25, 7, 4, 3, 1, 0)$$

Remark. The algorithm only calls `divmod`.
It would work for any "thing" that also has `divmod`.

Things to do when we see an algorithm:

- Is it correct?
- Is it fast?

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Theorem. For any $a, b \in \mathbb{N}_0$, the number e given by `euclid` is $\gcd(a, b)$.

Two things to prove:

- e divides a and b
- every d that divides a and b also divides e

Lemma. If d divides a and b, then d divides every element of `remainder`.

Proof by induction: property is true for r_0 and r_1

$$r_i = q \cdot r_{i+1} + r_{i+2}$$

divisible by d
$r_i = d \cdot x$

divisible by d
$(r_{i+1} = d \cdot y)$

So $r_{i+2} = d \cdot (x - qy)$ is divisible by d. □

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Theorem. For any $a, b \in \mathbb{N}_0$, the number e given by `euclid` is $\gcd(a, b)$.

Two things to prove:

- e divides a and b
- every d that divides a and b also divides e

Lemma. e divides a and b.

$$
\begin{aligned}
b &= q_2 \times a + r_2 && e \text{ divides } b \\
a &= q_3 \times r_2 + r_3 && e \text{ divides } a \\
r_2 &= q_4 \times r_3 + r_4 && e \text{ divides } r_2 \\
&\vdots \\
r_{k-2} &= q_k \times r_{k-1} + r_k && e \text{ divides } r_{k-2} \\
r_{k-1} &= q_{k+1} \times r_k + 0 && e \text{ divides } r_{k-1}
\end{aligned}
$$

this is e

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Runtime of Euclid's algorithm

How many times does the `while` loop run?

$$\texttt{remainders} = (25, 7, 4, 3, 1, 0)$$

Observation 1: the sequence is decreasing
Observation 2: $r_{i+2} < r_i/2$

If true: the sequence divides by 2 every 2 steps
$\rightsquigarrow 2\log_2(b)$ steps at most

```
[In [9]: mygcd(46368, 75025)
 Out[9]:
 [75025,
  46368,
  28657,
  17711,
  10946,
  6765,
  4181,
  2584,
  1597,
  987,
  610,
  377,
  233,
  144,
  89,
  55,
  34,
  21,
  13,
  8,
  5,
  3,
  2,
  1,
  0]
```

Runtime of Euclid's algorithm

How many times does the `while` loop run?

$$\texttt{remainders} = (25, 7, 4, 3, 1, 0)$$

Observation 1: the sequence is decreasing
Observation 2: $r_{i+2} < r_i/2$

If true: the sequence divides by 2 every 2 steps
$\rightsquigarrow 2\log_2(b)$ steps at most

Theorem. Let $(r_0, r_1, \ldots, r_k, 0)$ be the sequence of remainders computed by `euclid`.
Then $r_{i+2} < r_i/2$ holds for all $i \in \{0, \ldots, k-1\}$.

$$r_i = q \times r_{i+1} + r_{i+2} \text{ with } r_{i+2} \in \{0, \ldots, r_{i+1} - 1\}$$

```
def euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)

  return remainders[-2]
```

Theorem (Bézout). For all $a, b \in \mathbb{N}_0$, there exists $u, v \in \mathbb{Z}$ such that $u \cdot a + v \cdot b = \gcd(a, b)$.

Goal. Find $u, v \in \mathbb{Z}$ such that $25u + 7v = 1$.

$$
\begin{aligned}
25 &= 3 \times 7 + 4 \\
7 &= 1 \times 4 + 3 \\
4 &= 1 \times 3 + 1 \\
3 &= 3 \times 1 + 0 \\
\texttt{remainders} &= (25, 7, 4, 3, 1, 0)
\end{aligned}
$$

$$
\begin{aligned}
4 &= 25 - 3 \times 7 \\
3 &= 7 - 1 \times 4 \\
1 &= 4 - 1 \times 3 \\
&= 4 - 1 \times (7 - 1 \times 4) \\
&= -1 \times 7 + 2 \times 4 \\
&= -1 \times 7 + 2 \times (25 - 3 \times 7) \\
&= 2 \times 25 - 7 \times 7
\end{aligned}
$$

But.. why?

- Any common divisor of $\{7, 25\}$ must divide every number of the form $25u + 7v$.
- So if there exist u, v such that $25u + 7v = 1$, the gcd must be 1!
- We will soon want to compute the modular inverse of some numbers.
  Computing inverses = computing Bézout coefficients

Theorem (Bézout). For all $a, b \in \mathbb{N}_0$, there exists $u, v \in \mathbb{Z}$ such that $u \cdot a + v \cdot b = \gcd(a, b)$.

```
def extended_euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  u = [0,1]
  v = [1,0]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)
    u.append(u[-2] - q*u[-1])
    v.append(v[-2] - q*v[-1])

  return remainders[-2],u[-2],v[-2]
```

$$
\begin{aligned}
4 &= 25 - 3 \times 7 \\
3 &= 7 - 1 \times 4 \\
1 &= 4 - 1 \times 3 \\
&= 4 - 1 \times (7 - 1 \times 4) \\
&= -1 \times 7 + 2 \times 4 \\
&= -1 \times 7 + 2 \times (25 - 3 \times 7) \\
&= 2 \times 25 - 7 \times 7
\end{aligned}
$$

Theorem (Bézout). For all $a, b \in \mathbb{N}_0$, there exists $u, v \in \mathbb{Z}$ such that $u \cdot a + v \cdot b = \gcd(a, b)$.

```
def extended_euclid(a,b):
  if a > b:
    a,b = b,a # swap a and b
  if a == 0:
    return b

  remainders = [b,a]
  u = [0,1]
  v = [1,0]
  while remainders[-1] != 0:
    b = remainders[-2]
    a = remainders[-1]
    q,r = divmod(b,a)
    remainders.append(r)
    u.append(u[-2] - q*u[-1])
    v.append(v[-2] - q*v[-1])

  return remainders[-2],u[-2],v[-2]
```

$$25 = 1 \times 25 + 0 \times 7$$
$$7 = 0 \times 25 + 1 \times 7$$
$$4 = 25 - 3 \times 7$$
$$3 = -1 \times 25 + 4 \times 7$$
$$1 = 2 \times 25 - 7 \times 7$$

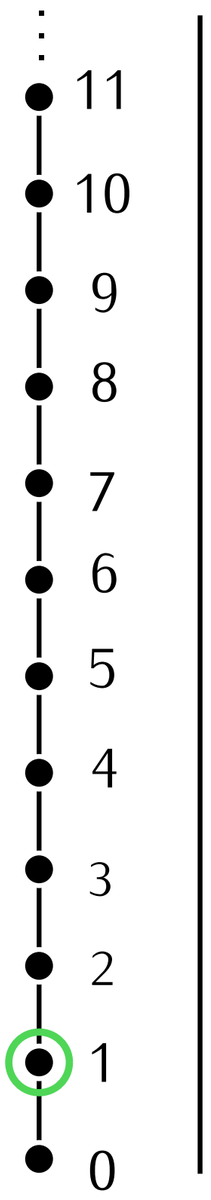
Invariant (property that remains true for every i):

$$r_i = u_i \cdot a + v_i \cdot b$$

$$R = \{(n, m) \in \mathbb{N}_0^2 \mid n \leq m\}$$

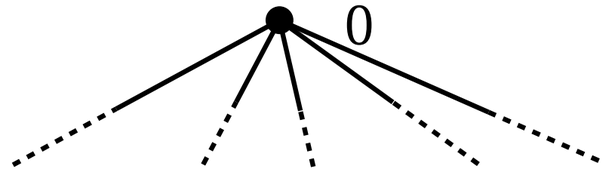
- Linear
- Minimal elements: 0
- Maximal elements: None
- $n \wedge m$ always exists: $\min(n, m)$
- $n \vee m$ always exists: $\max(n, m)$

Theorem. Every number n can be written (uniquely!) as $1 + 1 + 1 + \dots$

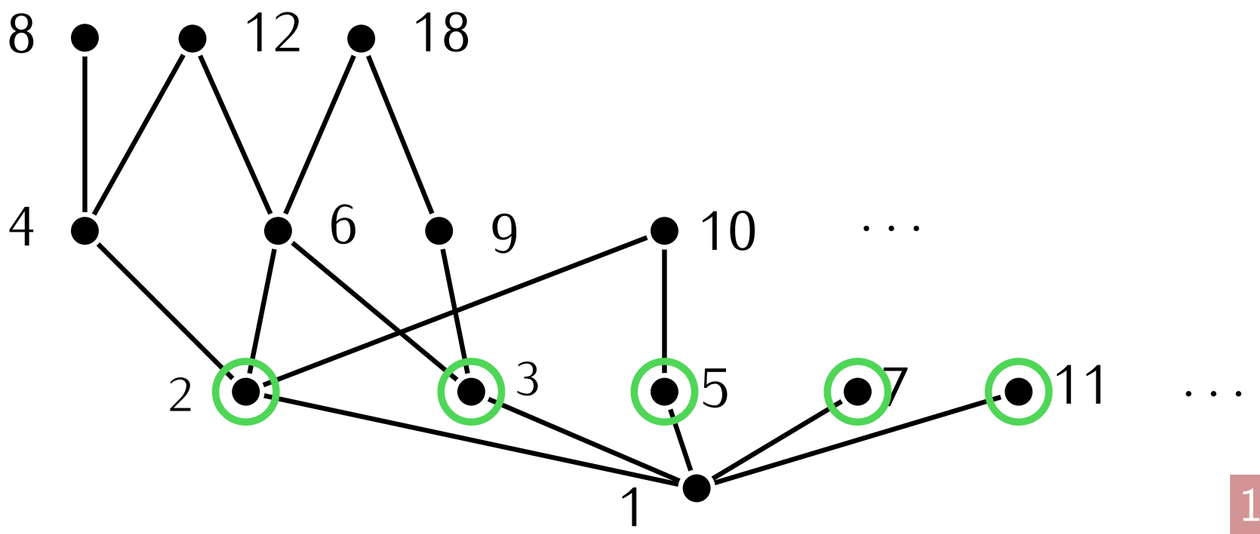


$$R = \{(n, m) \in \mathbb{N}_0^2 \mid n \text{ divides } m\}$$

- Not linear
- Minimal elements: 1
- Maximal elements: 0
- $n \wedge m$: gcd
- $n \vee m$: lcm

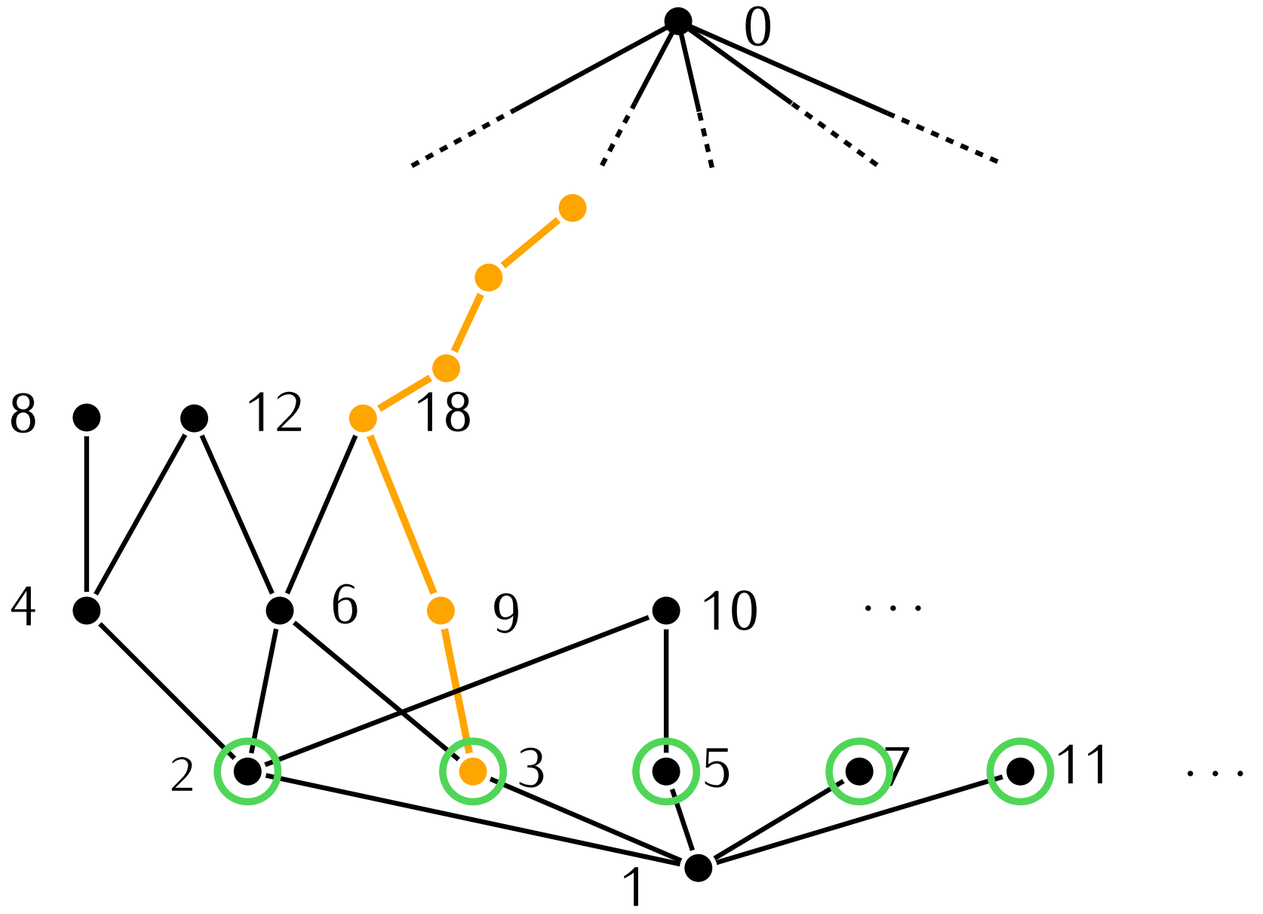


Theorem. Every number n can be written (uniquely!) as $2^{p_2} 3^{p_3} \dots$



Definition. A number $p \in \mathbb{N}_0$ is prime if it is > 1 and it is only divisible by 1 and itself.

Theorem. Every number $n \in \mathbb{N}$ that is not 1 is divisible by a prime number.

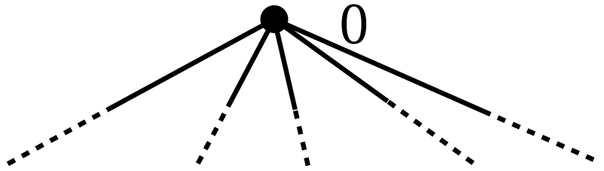


(*) Being prime can be defined for $p \in \mathbb{Z}$ as well

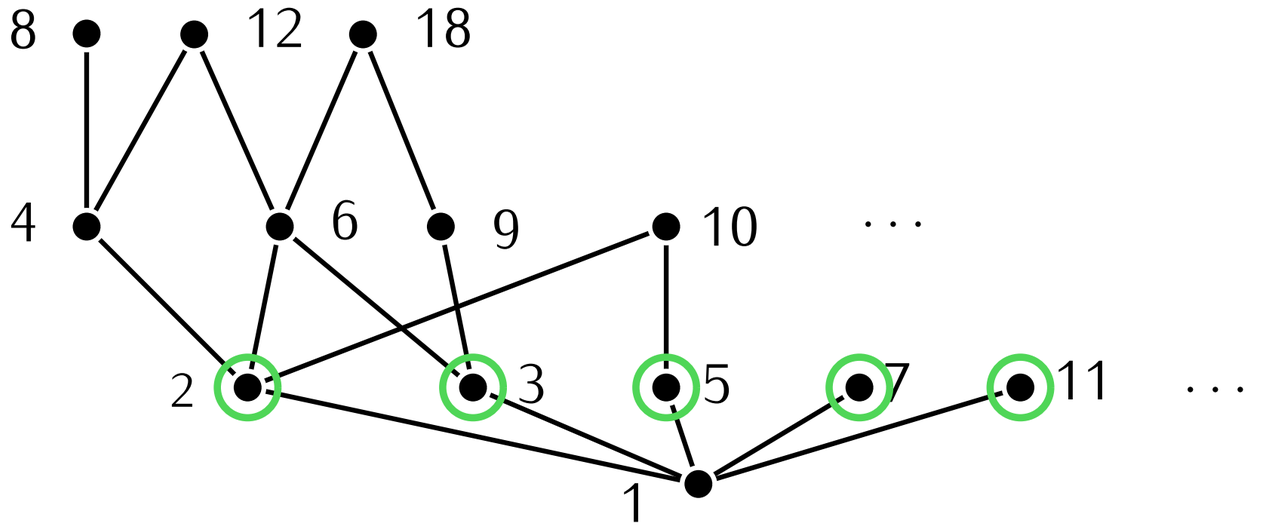
Definition. A number $p \in \mathbb{N}_0$ is prime if it is > 1 and it is only divisible by 1 and itself.

Theorem. Every number $n \in \mathbb{N}$ that is not 1 is divisible by a prime number.

Theorem. There are infinitely many prime numbers.



- Let $p_1, \ldots, p_k$ be any list of prime numbers.
- Define $N = p_1 \ldots p_k + 1$
- N is divisible by a prime p



(*) Being prime can be defined for $p \in \mathbb{Z}$ as well

Definition. A number $p \in \mathbb{N}_0$ is prime if it is > 1 and it is only divisible by 1 and itself.

Theorem. Every number $n \in \mathbb{N}$ that is not 1 is divisible by a prime number.

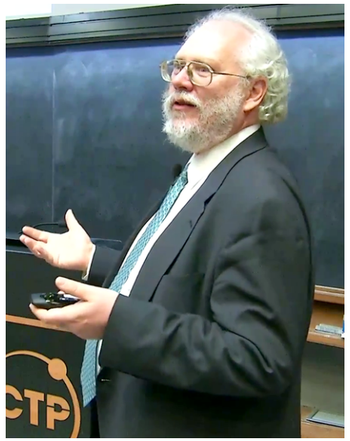
Theorem. There are infinitely many prime numbers.

- Let $p_1, \ldots, p_k$ be any list of prime numbers.
- Define $N = p_1 \ldots p_k + 1$
- N is divisible by a prime p
- $N = (p_1 \ldots p_{k-1})p_k + 1$ and $1 < p_k$, so p is not p_k
- Similarly for every $i \in \{1, \ldots, k-1\}$, N is not divisible by p_i.
  So $p \notin \{p_1, \ldots, p_k\}$
- $p_1, \ldots, p_k$ cannot be a list of all the primes $\square$

(*) Being prime can be defined for $p \in \mathbb{Z}$ as well

Theorem. Let $n \in \mathbb{N}$ be such that $n \geq 2$.
There exist prime numbers $p_1 < \cdots < p_k$ and $e_1, \ldots, e_k \in \mathbb{N}$ such that $n = p_1^{e_1} \cdots p_k^{e_k}$.
Moreover this decomposition is unique.

- Important in math (building blocks for all numbers)
- Important in crypto (RSA)
- It seems hard to compute $p_1, \ldots, p_k$ if given $n \in \mathbb{N}$
- **Theoretically,** there is a quantum algorithm that can do this and **break RSA**.



Peter Shor