KNAPSACK Is NP-Complete\textsuperscript{a}

- KNAPSACK $\in$ NP: Guess an $S$ and check the constraints.
- We shall reduce EXACT COVER BY 3-SETS to KNAPSACK, in which $v_i = w_i$ for all $i$ and $K = W$.
- The simplified KNAPSACK now asks if a subset of $v_1, v_2, \ldots, v_n$ adds up to exactly $K$.\textsuperscript{b}
  - Picture yourself as a radio DJ.

\textsuperscript{a}Karp (1972).

\textsuperscript{b}This problem is called SUBSET SUM.
The Proof (continued)

• The primary differences between the two problems are:
  
  – Sets vs. numbers.
  
  – Union vs. addition.

• We are given a family $F = \{S_1, S_2, \ldots, S_n\}$ of size-3 subsets of $U = \{1, 2, \ldots, 3m\}$.

• EXACT COVER BY 3-SETS asks if there are $m$ disjoint sets in $F$ that cover the set $U$.

\(^a\)Thanks to a lively class discussion on November 16, 2010.
The Proof (continued)

- Think of a set as a bit vector in \( \{0, 1\}^{3m} \).
  - Assume \( m = 3 \).
  - 110010000 means the set \( \{1, 2, 5\} \).
  - 001100010 means the set \( \{3, 4, 8\} \).
- Assume there are \( n = 5 \) size-3 subsets in \( F \).
- Our goal is

\[
\underbrace{11 \ldots 1}_{3m}.
\]
The Proof (continued)

- A bit vector can also be seen as a binary *number*.
- Set union resembles addition:

\[
\begin{array}{c}
001100010 \\
+ \quad 110010000 \\
\hline
111110010
\end{array}
\]

which denotes the set \(\{1, 2, 3, 4, 5, 8\}\), as desired.
The Proof (continued)

- Trouble occurs when there is carry:

\[
\begin{array}{c}
010000000 \\
+ 010000000 \\
\hline
100000000
\end{array}
\]

which denotes the set \{1\}, not the desired \{2\}. 
The Proof (continued)

• Or consider

\[ \begin{array}{c}
001100010 \\
+ \ 001110000 \\
\hline
011010010
\end{array} \]

which denotes the set \( \{2, 3, 5, 8\} \), not the desired \( \{3, 4, 5, 8\}\).\(^a\)

\(^a\)Corrected by Mr. Chihwei Lin (D97922003) on January 21, 2010.
The Proof (continued)

- Carry may also lead to a situation where we obtain our solution $1 \ 1 \cdots 1$ with more than $m$ sets in $F$.

- For example,

\[
\begin{align*}
&000100010 \\
&001110000 \\
&101100000 \\
&000001101 \\
&1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\
\end{align*}
\]

- But the correct answer, $\{1, 3, 4, 5, 6, 7, 8, 9\}$, is not an exact cover.
The Proof (continued)

- And it uses 4 sets instead of the required $m = 3$.\(^a\)

- To fix this problem, we enlarge the base just enough so that there are no carries.\(^b\)

- Because there are $n$ vectors in total, we change the base from 2 to $n + 1$.

\(^a\)Thanks to a lively class discussion on November 20, 2002.

\(^b\)You cannot map $\cup$ to $\lor$ because KNAPSACK requires $\plus$. 
The Proof (continued)

- Set $v_i$ to be the integer corresponding to the bit vector encoding $S_i$ in base $n + 1$:

$$v_i = \sum_{j \in S_i} 1 \times (n + 1)^{3m-j}$$ (4)

- Set

$$K = \sum_{j=0}^{3m-1} 1 \times (n + 1)^j = \overbrace{11\cdots1}^{3m} \quad \text{(base } n + 1).$$

- Now in base $n + 1$, if there is a set $S$ such that

$$\sum_{i \in S} v_i = \overbrace{11\cdots1}^{3m},$$
then every position must be contributed by exactly one $v_i$ and $|S| = m$. 
The Proof (continued)

- For example, the case on p. 420 becomes

\[
000100010 \\
001110000 \\
101100000 \\
+ \quad 000001101 \\
\hline
102311111
\]

\[
\text{in base } n + 1 = 6.
\]

- It does not meet the goal.
The Proof (continued)

• Suppose $F$ admits an exact cover, say $\{S_1, S_2, \ldots, S_m\}$.

• Then picking $S = \{1, 2, \ldots, m\}$ clearly results in

$$v_1 + v_2 + \cdots + v_m = 11\cdots1.$$  

• It is important to note that the meaning of addition (+) is independent of the base.\(^a\)
  
  – It is just regular addition.
  
  – But an $S_i$ may give rise to different integer $v_i$’s in Eq. (4) on p. 422 under different bases.

\(^a\)Contributed by Mr. Kuan-Yu Chen (R92922047) on November 3, 2004.
The Proof (concluded)

- On the other hand, suppose there exists an \( S \) such that

\[
\sum_{i \in S} v_i = \underbrace{11 \cdots 1}_{3m}
\]

in base \( n + 1 \).

- The no-carry property implies that \(|S| = m\) and

\( \{S_i : i \in S\} \)

is an exact cover.
An Example

- Let $m = 3$, $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and

  $S_1 = \{1, 3, 4\}$,
  $S_2 = \{2, 3, 4\}$,
  $S_3 = \{2, 5, 6\}$,
  $S_4 = \{6, 7, 8\}$,
  $S_5 = \{7, 8, 9\}$.

- Note that $n = 5$, as there are 5 $S_i$’s.
An Example (continued)

- Our reduction produces

\[
K = \sum_{j=0}^{3 \times 3 - 1} 6^j = \overbrace{11 \cdots 1}^{3 \times 3} \quad \text{(base 6)} = 2015539,
\]

\[
v_1 = 101100000 = 1734048,
\]

\[
v_2 = 011100000 = 334368,
\]

\[
v_3 = 010011000 = 281448,
\]

\[
v_4 = 000001110 = 258,
\]

\[
v_5 = 000000111 = 43.
\]
An Example (concluded)

• Note $v_1 + v_3 + v_5 = K$ because

\[
\begin{array}{c}
101100000 \\
010011000 \\
\hline
+ 000000111 \\
\hline
111111111
\end{array}
\]

• Indeed,

\[S_1 \cup S_3 \cup S_5 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\},\]

an exact cover by 3-sets.
BIN PACKING

- We are given \( N \) positive integers \( a_1, a_2, \ldots, a_N \), an integer \( C \) (the capacity), and an integer \( B \) (the number of bins).

- BIN PACKING asks if these numbers can be partitioned into \( B \) subsets, each of which has total sum at most \( C \).

- Think of packing bags at the check-out counter.

**Theorem 47** BIN PACKING is NP-complete.
BIN PACKING (concluded)

- But suppose $a_1, a_2, \ldots, a_N$ are randomly distributed between 0 and 1.

- Let $B$ be the smallest number of unit-capacity bins capable of holding them.

- Then $B$ can differ from its average by more than $t$ with probability at most $2e^{-2t^2/N}$.\(^a\)

\(^a\)Dubhashi and Panconesi (2012).
INTEGER PROGRAMMING

- INTEGER PROGRAMMING asks whether a system of linear inequalities with integer coefficients has an integer solution.

- In contrast, LINEAR PROGRAMMING asks whether a system of linear inequalities with integer coefficients has a rational solution.
INTEGRER PROGRAMMING Is NP-Complete

- SET COVERING can be expressed by the inequalities
  \[ Ax \geq \vec{1}, \sum_{i=1}^{n} x_i \leq B, \ 0 \leq x_i \leq 1, \]
  where
  - \( x_i \) is one if and only if \( S_i \) is in the cover.
  - \( A \) is the matrix whose columns are the bit vectors of the sets \( S_1, S_2, \ldots \)
  - \( \vec{1} \) is the vector of 1s.
  - The operations in \( Ax \) are standard matrix operations.

- This shows INTEGER PROGRAMMING is NP-hard.

- Many NP-complete problems can be expressed as an INTEGER PROGRAMMING problem.

\(^a\)Karp (1972).
Easier or Harder?\textsuperscript{a}

- Adding restrictions on the allowable *problem instances* will not make a problem harder.
  - We are now solving a subset of problem instances or special cases.
  - The INDEPENDENT SET proof (p. 361) and the KNAPSACK proof (p. 414).
  - SAT to 2SAT (easier by p. 342).
  - CIRCUIT VALUE to MONOTONE CIRCUIT VALUE (equally hard by p. 314).

\textsuperscript{a}Thanks to a lively class discussion on October 29, 2003.
Easier or Harder? (concluded)

- Adding restrictions on the allowable solutions (the solution space) may make a problem harder, equally hard, or easier.

- It is problem dependent.
  - MIN CUT to BISECTION WIDTH (harder by p. 389).
  - LINEAR PROGRAMMING to INTEGER PROGRAMMING (harder by p. 431).
  - SAT to NAESAT (equally hard by p. 355) and MAX CUT to MAX BISECTION (equally hard by p. 387).
  - 3-COLORING to 2-COLORING (easier by p. 398).
coNP and Function Problems
coNP

- NP is the class of problems that have succinct certificates (recall Proposition 36 on p. 326).

- By definition, coNP is the class of problems whose complement is in NP.

- coNP is therefore the class of problems that have succinct disqualifications:
  - A “no” instance of a problem in coNP possesses a short proof of its being a “no” instance.
  - Only “no” instances have such proofs.
coNP (continued)

- Suppose $L$ is a coNP problem.
- There exists a polynomial-time nondeterministic algorithm $M$ such that:
  - If $x \in L$, then $M(x) = \text{"yes"}$ for all computation paths.
  - If $x \notin L$, then $M(x) = \text{"no"}$ for some computation path.
- Note that if we swap "yes" and "no" of $M$, the new algorithm $M'$ decides $\overline{L} \in \text{NP}$ in the classic sense (p. 109).
\[ x \in L \]

\[ x \notin L \]
So there are 3 major approaches to proving $L \in \text{coNP}$.

1. Prove $\overline{L} \in \text{NP}$.
2. Prove that “no” instances possess short proofs.
3. Write an algorithm for it.
coNP (concluded)

- Clearly $P \subseteq \text{coNP}$.
- It is not known if

$$P = \text{NP} \cap \text{coNP}.$$ 

- Contrast this with

$$R = \text{RE} \cap \text{coRE}$$

(see Proposition 12 on p. 170).
Some coNP Problems

• VALIDITY ∈ coNP.
  – If \( \phi \) is not valid, it can be disqualified very succinctly: a truth assignment that does not satisfy it.

• SAT COMPLEMENT ∈ coNP.
  – SAT COMPLEMENT is the complement of SAT.
  – The disqualification is a truth assignment that satisfies it.

• HAMILTONIAN PATH COMPLEMENT ∈ coNP.
  – The disqualification is a Hamiltonian path.
Some coNP Problems (concluded)

- **OPTIMAL TSP (D) ∈ coNP.**
  
  - OPTIMAL TSP (D) asks if the optimal tour has a total distance of $B$, where $B$ is an input.$^\text{a}$
  
  - The disqualification is a tour with a length $< B$.

---

$^\text{a}$Defined by Mr. Che-Wei Chang (R95922093) on September 27, 2006.
A Nondeterministic Algorithm for SAT COMPLEMENT

\( \phi \) is a boolean formula with \( n \) variables.

1: \textbf{for} \( i = 1, 2, \ldots, n \) \textbf{do} \\
2: \quad \text{Guess} \( x_i \in \{0, 1\}; \{\text{Nondeterministic choice.}\} \) \\
3: \textbf{end for} \\
4: \{\text{Verification:}\} \\
5: \quad \textbf{if} \ \phi(x_1, x_2, \ldots, x_n) = 1 \ \textbf{then} \\
6: \quad \quad \text{“no”;} \\
7: \quad \textbf{else} \\
8: \quad \quad \text{“yes”;} \\
9: \quad \textbf{end if}
Analysis

- The algorithm decides language \( \{ \phi : \phi \text{ is unsatisfiable} \} \).
  - The computation tree is a complete binary tree of depth \( n \).
  - Every computation path corresponds to a particular truth assignment out of \( 2^n \).
  - \( \phi \) is unsatisfiable iff every truth assignment falsifies \( \phi \).
  - But every truth assignment falsifies \( \phi \) iff every computation path results in “yes.”
An Alternative Characterization of coNP

**Proposition 48** Let $L \subseteq \Sigma^*$ be a language. Then $L \in \text{coNP}$ if and only if there is a polynomially decidable and polynomially balanced relation $R$ such that

$$L = \{ x : \forall y (x, y) \in R \}.$$  

(As on p. 325, we assume $|y| \leq |x|^k$ for some $k$.)

- $\bar{L} = \{ x : \exists y (x, y) \in \neg R \}$.

- Because $\neg R$ remains polynomially balanced, $\bar{L} \in \text{NP}$ by Proposition 36 (p. 326).

- Hence $L \in \text{coNP}$ by definition.
coNP-Completeness

Proposition 49  \( L \) is NP-complete if and only if its complement \( \overline{L} = \Sigma^* - L \) is coNP-complete.

Proof (\( \Rightarrow \); the \( \Leftarrow \) part is symmetric)

- Let \( \overline{L}' \) be any coNP language.
- Hence \( L' \in \text{NP} \).
- Let \( R \) be the reduction from \( L' \) to \( L \).
- So \( x \in L' \) if and only if \( R(x) \in L \).
- Equivalently, \( x \not\in L' \) if and only if \( R(x) \not\in L \) (the law of transposition).
coNP Completeness (concluded)

- So $x \in \bar{L}'$ if and only if $R(x) \in \bar{L}$.
- $R$ is a reduction from $\bar{L}'$ to $\bar{L}$.
- This shows $\bar{L}$ is coNP-hard.
- But $\bar{L} \in \text{coNP}$.
- This shows $\bar{L}$ is coNP-complete.
Some coNP-Complete Problems

- SAT COMPLEMENT is coNP-complete.
- VALIDITY is coNP-complete.
  - $\phi$ is valid if and only if $\neg\phi$ is not satisfiable.
  - The reduction from SAT COMPLEMENT to VALIDITY is hence easy.
- HAMILTONIAN PATH COMPLEMENT is coNP-complete.
Possible Relations between P, NP, coNP

1. $P = NP = coNP$.

2. $NP = coNP$ but $P \neq NP$.

3. $NP \neq coNP$ and $P \neq NP$.

- This is the current “consensus.”

---

\(^a\)Carl Gauss (1777–1855), “I could easily lay down a multitude of such propositions, which one could neither prove nor dispose of.”
The Primality Problem

- An integer $p$ is **prime** if $p > 1$ and all positive numbers other than 1 and $p$ itself cannot divide it.

- PRIMES asks if an integer $N$ is a prime number.

- Dividing $N$ by $2, 3, \ldots, \sqrt{N}$ is *not* efficient.
  - The length of $N$ is only $\log N$, but $\sqrt{N} = 2^{0.5 \log N}$.
  - So it is an exponential-time algorithm.

- A polynomial-time algorithm for PRIMES was not found until 2002 by Agrawal, Kayal, and Saxena!

- Later, we will focus on efficient “probabilistic” algorithms for PRIMES (used in *Mathematica*, e.g.).
1: if $n = a^b$ for some $a, b > 1$ then
2:     return “composite”;
3: end if
4: for $r = 2, 3, \ldots, n - 1$ do
5:     if $\gcd(n, r) > 1$ then
6:         return “composite”;
7:     end if
8:     if $r$ is a prime then
9:         Let $q$ be the largest prime factor of $r - 1$;
10:        if $q \geq 4\sqrt{r} \log n$ and $n^{(r-1)/q} \not\equiv 1 \mod r$ then
11:            break; {Exit the for-loop.}
12:        end if
13:     end if
14: end for {r - 1 has a prime factor $q \geq 4\sqrt{r} \log n$.}
15: for $a = 1, 2, \ldots, 2\sqrt{r} \log n$ do
16:     if $(x - a)^n \not\equiv (x^n - a) \mod (x^r - 1)$ in $\mathbb{Z}_n[x]$ then
17:         return “composite”;
18:     end if
19: end for
20: return “prime”; {The only place with “prime” output.}
The Primality Problem (concluded)

- NP ∩ coNP is the class of problems that have succinct certificates and succinct disqualifications.
  - Each “yes” instance has a succinct certificate.
  - Each “no” instance has a succinct disqualification.
  - No instances have both.

- We will see that PRIMES ∈ NP ∩ coNP.
  - In fact, PRIMES ∈ P as mentioned earlier.
Primitive Roots in Finite Fields

Theorem 50 (Lucas and Lehmer (1927)) A number $p > 1$ is a prime if and only if there is a number $1 < r < p$ such that

1. $r^{p-1} = 1 \mod p$, and
2. $r^{(p-1)/q} \neq 1 \mod p$ for all prime divisors $q$ of $p-1$.

- This $r$ is called the **primitive root** or **generator**.
- We will prove the theorem later (see pp. 464ff).

---

Derrick Lehmer (1905–1991)
Pratt’s Theorem

Theorem 51 (Pratt (1975)) \( \text{PRIMES} \in NP \cap \text{coNP} \).

- \( \text{PRIMES} \) is in \( \text{coNP} \) because a succinct disqualification is a proper divisor.
  - A proper divisor of a number \( n \) means \( n \) is not a prime.
- Now suppose \( p \) is a prime.
- \( p \)'s certificate includes the \( r \) in Theorem 50 (p. 453).
- Use recursive doubling to check if \( r^{p-1} = 1 \mod p \) in time polynomial in the length of the input, \( \log_2 p \).
  - \( r, r^2, r^4, \ldots \mod p \), a total of \( \sim \log_2 p \) steps.
The Proof (concluded)

- We also need all prime divisors of $p - 1$: $q_1, q_2, \ldots, q_k$.
  - Whether $r, q_1, \ldots, q_k$ are easy to find is irrelevant.
  - There may be multiple choices for $r$.
- Checking $r^{(p-1)/q_i} \neq 1 \mod p$ is also easy.
- Checking $q_1, q_2, \ldots, q_k$ are all the divisors of $p - 1$ is easy.
- We still need certificates for the primality of the $q_i$’s.
- The complete certificate is recursive and tree-like:
  \[
  C(p) = (r; q_1, C(q_1), q_2, C(q_2), \ldots, q_k, C(q_k)).
  \]
- We next prove that $C(p)$ is succinct.
- As a result, $C(p)$ can be checked in polynomial time.

\[\text{©2014 Prof. Yuh-Dauh Lyuu, National Taiwan University} \]
The Succinctness of the Certificate

**Lemma 52** The length of $C(p)$ is at most quadratic at $5 \log_2^2 p$.

- This claim holds when $p = 2$ or $p = 3$.
- In general, $p - 1$ has $k \leq \log_2 p$ prime divisors $q_1 = 2, q_2, \ldots, q_k$.
  - Reason:
    \[
    2^k \leq \prod_{i=1}^{k} q_i \leq p - 1.
    \]
- Note also that, as $q_1 = 2$,
  \[
  \prod_{i=2}^{k} q_i \leq \frac{p - 1}{2}. \tag{5}
  \]
The Proof (continued)

- $C(p)$ requires:
  - 2 parentheses;
  - $2k < 2 \log_2 p$ separators (at most $2 \log_2 p$ bits);
  - $r$ (at most $\log_2 p$ bits);
  - $q_1 = 2$ and its certificate 1 (at most 5 bits);
  - $q_2, \ldots, q_k$ (at most $2 \log_2 p$ bits);\(^a\)
  - $C(q_2), \ldots, C(q_k)$.

\(^a\)Why?
The Proof (concluded)

- $C(p)$ is succinct because, by induction,

$$|C(p)| \leq 5 \log_2 p + 5 + 5 \sum_{i=2}^{k} \log_2 q_i$$

$$\leq 5 \log_2 p + 5 + 5 \left( \sum_{i=2}^{k} \log_2 q_i \right)^2$$

$$\leq 5 \log_2 p + 5 + 5 \log_2 p - \frac{1}{2} \text{ by inequality (5)}$$

$$< 5 \log_2 p + 5 + 5(\log_2 p - 1)^2$$

$$= 5 \log_2^2 p + 10 - 5 \log_2 p \leq 5 \log_2^2 p$$

for $p \geq 4$. 
A Certificate for 23\(^a\)

- Note that 7 is a primitive root modulo 23 and 
  \(23 - 1 = 22 = 2 \times 11\).

- So
  
  \[ C(23) = (7; 2, C(2), 11, C(11)). \]

- Note that 2 is a primitive root modulo 11 and 
  \(11 - 1 = 10 = 2 \times 5\).

- So
  
  \[ C(11) = (2; 2, C(2), 5, C(5)). \]

\(^a\)Thanks to a lively discussion on April 24, 2008.
A Certificate for 23 (concluded)

- Note that 2 is a primitive root modulo 5 and $5 - 1 = 4 = 2^2$.

- So

$$C(5) = (2; 2, C(2)).$$

- In summary,

$$C(23) = (7; 2, C(2), 11, (2; 2, C(2), 5, (2; 2, C(2))))).$$
Basic Modular Arithmetics

- Let $m, n \in \mathbb{Z}^+$.
- $m \mid n$ means $m$ divides $n$; $m$ is $n$’s divisor.
- We call the numbers $0, 1, \ldots, n - 1$ the residue modulo $n$.
- The greatest common divisor of $m$ and $n$ is denoted $\gcd(m, n)$.
- The $r$ in Theorem 50 (p. 453) is a primitive root of $p$.
- We now prove the existence of primitive roots and then Theorem 50 (p. 453).

---

$a$Carl Friedrich Gauss.
Basic Modular Arithmetics (concluded)

• We use

\[ a \equiv b \mod n \]

if \( n \mid (a - b) \).

– So \( 25 \equiv 38 \mod 13 \).

• We use

\[ a = b \mod n \]

if \( b \) is the remainder of \( a \) divided by \( n \).

– So \( 25 = 12 \mod 13 \).
Euler’s\textsuperscript{a} Totient or Phi Function

- Let
  \[ \Phi(n) = \{m : 1 \leq m < n, \gcd(m, n) = 1\} \]
  be the set of all positive integers less than \( n \) that are prime to \( n \).\textsuperscript{b}
  - \( \Phi(12) = \{1, 5, 7, 11\} \).

- Define \textbf{Euler’s function} of \( n \) to be \( \phi(n) = |\Phi(n)| \).

- \( \phi(p) = p - 1 \) for prime \( p \), and \( \phi(1) = 1 \) by convention.

- Euler’s function is not expected to be easy to compute without knowing \( n \)’s factorization.

\textsuperscript{a}Leonhard Euler (1707–1783).
\textsuperscript{b}\( Z_n^* \) is an alternative notation.
Two Properties of Euler’s Function

The inclusion-exclusion principle\(^a\) can be used to prove the following.

**Lemma 53** \(\phi(n) = n \prod_{p|n} (1 - \frac{1}{p}).\)

- If \(n = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}\) is the prime factorization of \(n\), then

\[
\phi(n) = n \prod_{i=1}^{\ell} \left(1 - \frac{1}{p_i}\right).
\]

**Corollary 54** \(\phi(mn) = \phi(m) \phi(n)\) if \(\gcd(m, n) = 1.\)

\(^a\)Consult any textbook on discrete mathematics.
A Key Lemma

Lemma 55 $\sum_{m|n} \phi(m) = n$.

- Let $n = \prod_{i=1}^{\ell} p_i^{k_i}$ be the prime factorization of $n$ and consider

$$\prod_{i=1}^{\ell} \left[ \phi(1) + \phi(p_i) + \cdots + \phi(p_i^{k_i}) \right].$$

(6)

- Equation (6) equals $n$ because $\phi(p_i^{k_i}) = p_i^{k_i} - p_i^{k_i-1}$ by Lemma 53 (p. 466) so $\phi(1) + \phi(p_i) + \cdots + \phi(p_i^{k_i}) = p_i^{k_i}$.

- Expand Eq. (6) to yield

$$n = \sum_{k_1', \ldots, k_\ell' \leq k_\ell} \prod_{i=1}^{\ell} \phi(p_i^{k_i'}).$$
The Proof (concluded)

- By Corollary 54 (p. 466),
  \[ \prod_{i=1}^{\ell} \phi(p_i^{k_i'}) = \phi \left( \prod_{i=1}^{\ell} p_i^{k_i'} \right). \]

- So Eq. (6) becomes
  \[ n = \sum_{k_1' \leq k_1, \ldots, k_\ell' \leq k_\ell} \phi \left( \prod_{i=1}^{\ell} p_i^{k_i'} \right). \]

- Each \( \prod_{i=1}^{\ell} p_i^{k_i'} \) is a unique divisor of \( n = \prod_{i=1}^{\ell} p_i^{k_i} \).

- Equation (6) becomes
  \[ \sum_{m \mid n} \phi(m). \]
Leonhard Euler (1707–1783)