

## Some “difficult” problems

1. Hyphenation (dk: “orddeling”, f.eks. lak-ridser).
2. Decide if a program halts – the **Halting Problem**.
3. Decide if a Boolean expression is can be true – the **Satisfiability Problem**.
4. The Towers of Hanoi.

## Problem 4: Tower of Hanoi



- Let  $T_n$  denote the minimum number of moves to move  $n$  disks from one peg to another.
- We have  $T_0 = 0, T_1 = 1, T_2 = 3, \dots$
- In general

$$T_n = 2T_{n-1} + 1 \quad \text{for } n > 0.$$

- Solve the recurrence:

$$T_n = 2^n - 1.$$

## (Informal) Definition of P and NP

- The complexity class P is the set of problems that can be **solved** in polynomial time.
- The complexity class NP is the set of problems whose solution can be **checked** in polynomial time.

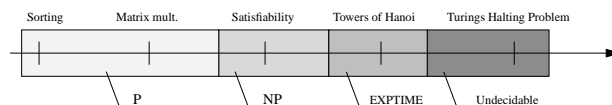
## Problem 2: Turings Halting Problem

- Assume “**Terminate**( $A$ )” returns true if and only if program  $A$  terminates.
- Consider the program  $B$ :

$B(C) =$  if **Terminate**( $C$ ) then loop forever  
else stop

- Then run  $B(B)$ ...
  - If **Terminate**( $B$ ) is true, then  $B$  loops forever
  - If **Terminate**( $B$ ) is false, then  $B$  stops.

## Overview of “problems”



## Examples of problems in NP

- **Satisfiability of Boolean Expressions:**

**Given:** A set of variables  $U$  and a Boolean expression  $E$  over  $U$ .

**Question:** Is there a truth assignment for  $U$  that satisfies  $E$ ?

- **Hamiltonian cycle:**

**Given:** An undirected graph  $G = (V, E)$ .

**Question:** Does  $G$  have a simple cycle that contains each vertex in  $V$ ?

## Is P = NP ?

- This is one of the greatest problems in theoretical computer science over the last 25 years.
- Today it is *believed* that  $P \neq NP$ .
- The theory of NP-completeness is a good indication.

## What is a “Problem”?

- Abstract problem  $Q$  is a binary relation on a set  $I$  of *instances* and a set  $S$  of *solutions*.
- Example: for  $Q = \text{SHORTEST-PATH}$ 
  - $I$  is a set of triples  $(G, u, v)$
  - $S$  is a sequence of vertices.
- Since a solution to SHORTEST-PATH is not unique, for a given  $I$ ,  $Q$  may contain several pairs  $(I, S)$ .

## Decision Problems

- **Decision problems** have yes/no answers, corresponding to  $S = \{0, 1\}$ .
- Example: PATH is a decision problem where
  - $I$  is a set of quadruples  $(G, u, v, k)$
  - $S = 1$  (yes) if a shortest path from  $u$  to  $v$  has length at most  $k$ .
- Note: The two examples of problems in NP were in fact decision problems.

## Concrete problems: Encodings

- **Encoding** of  $S$  is a mapping  $e : S \rightarrow \{0, 1\}^*$ .
- E.g.,  $e(\mathbb{A}) = 100001$  in ASCII.
- A **concrete problem** is the encoding of an abstract problem.
- An algorithm **solves** a concrete problem in time  $O(T(n))$  if, for any instance  $i \in I$ , it solves  $i$  in at most  $O(T(n))$  timer where  $n = |i|$ .

## Concrete problems: Encodings (2)

- Note: runtime depends on encoding !
- Example: consider an algorithm that takes a number  $k$  as input and runs in time  $\Theta(k)$ .
  - If  $k$  is encoded in unary, the runtime is  $O(n)$  on  $n$ -length inputs.
  - If  $k$  is encoded in binary, the runtime is  $O(2^n)$  since  $n = \lceil \lg k \rceil$ .
- Lemma 36.1: an encoding in any base larger than 1 is OK.
- $\langle Q \rangle$  denotes an encoding of  $Q$ .

## Formal languages

- **Alphabet**  $\Sigma$ : finite set of symbols, e.g.,  $\Sigma = \{0, 1\}$ .
- **Language** over  $\Sigma$ : a set of strings, e.g.,
$$L = \{10, 11, 101, 111, 1011, 1101, 10001, \dots\}.$$
- Operations:

<b>Empty string</b>	$\epsilon$
<b>Empty language:</b>	$\emptyset$
<b>Union:</b>	$L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$
<b>Intersection:</b>	$L_1 \cap L_2 = \{x : x \in L_1 \text{ and } x \in L_2\}$
<b>Complement:</b>	$\bar{L} = \Sigma^* - L$
<b>Concatenation:</b>	$L_1 L_2 = \{x_1 x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}$
<b>Closure:</b>	$L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \dots$

## Problems as formal languages

- Decision problem  $Q$  as a formal language  $L$ :

$$L = \{x \in \Sigma^* : Q(x) = 1\}.$$

- For example

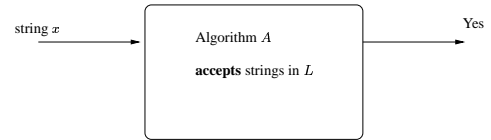
$$\text{PATH} = \{\langle G, u, v, k \rangle : \dots \text{and there exists a path from } u \text{ to } v \text{ in } G \text{ with length at most } k\}.$$

## Algorithms as formal languages

- “Algorithm  $A$  solves decision problem  $Q$  in polynomial time”:

There exists a  $k$  such that for each instance  $I$ ,  $A$  checks whether  $\langle I \rangle \in L$  in time  $O(n^k)$  where  $n = |\langle I \rangle|$

- Algorithm  $A$  **accepts** a string  $x$  if, given input  $x$ , the algorithm outputs  $A(x) = 1$ .



## Algorithms as formal languages (2)

- Algorithm  $A$  **rejects** a string  $x$  if, given input  $x$ , the algorithm outputs  $A(x) = 0$ .

- Language **accepted** by  $A$ :

$$L = \{x \in \{0, 1\}^* : A(x) = 1\}$$

- Language  $L$  is **decided** by  $A$  if for all  $x \in \{0, 1\}^*$ ,  $A$  either accepts or rejects  $x$ .

- New definition of P:

$$P = \{L \subseteq \{0, 1\}^* : \text{there exists an algorithm } A \text{ that decides } L \text{ in polynomial time}\}.$$

## Summary

- Problem *instances* are encoded as *strings*.
- Problems are represented as *languages*.
- Algorithms are *string acceptors*.

## Sorting is in P

- Sorting integers as a decision problem:

$$\text{SORTED} = \{\langle \sigma \rangle : \sigma \text{ is a sequence of integers in ascending order}\}.$$

- For example,  $\sigma_1 = 2, 7, 19 \in \text{SORTED}$ , while  $\sigma_2 = 2, 7, 5 \notin \text{SORTED}$ .

- One encoding of  $\sigma_1$  is “2 , 7 , 1 9” = 50 44 55 44 49 57 = 00110010 00101100 ...

- Theorem:**  $\text{SORTED} \in P$ .

**Proof:** Need to show that there exists an algorithm  $A$  that decides SORTED in polynomial time.

That is, for each instance  $x \subseteq \{0, 1\}^*$ ,  $A$  needs to decide whether  $x \in \text{SORTED}$  in time  $O(n^k)$  where  $n = |x|$ .

## Verifying a solution

- For some problems, it is easier to check a solution than to find it!
- Example: the hamiltonian-cycle problem:

$$\text{HAM-CYCLE} = \{\langle G \rangle : G \text{ has a hamiltonian cycle}\}.$$

- Algorithm to decide HAM-CYCLE ?
- Easier problem: Given  $G$  and a sequence of vertices  $c$ , check that  $c$  is a hamiltonian cycle.

## Verifying a solution (2)

- Verification algorithm:  $A(x, y)$  ( $x$ : problem,  $y$ : proof/certificate)
- $A$  **verifies**  $x$  if there exists a  $y$  such that  $A(x, y) = 1$ .
- The **language verified by**  $A$  is
 
$$L = \{x \in \{0, 1\}^* : \text{there exists } y \in \{0, 1\}^* \text{ such that } A(x, y) = 1\}$$

## Definition of NP

- The complexity class NP is the set of languages that can be verified by a polynomial time algorithm:
 
$$\text{NP} = \{L \subseteq \{0, 1\}^* : \text{there exists a constant } c, \text{ a certificate } y \text{ with } |y| = O(|x|^c), \text{ and an algorithm } A(x, y) \text{ such that } A(x, y) = 1\}.$$
- Clearly,  $\text{HAM-CYCLE} \in \text{NP}$ .
- Note:  $L \in \text{P} \Rightarrow L \in \text{NP}$  (that is,  $\text{P} \subseteq \text{NP}$ ).

## NP-completeness

- **NP-complete problems**: if any one could be solved in polynomial time, then every problem in NP has a polynomial-time solution (that is,  $\text{P} = \text{NP}$ ).
- Does such a problem exist?
- It turns out that for example HAM-CYCLE is NP-complete.

## Reducibility

- Reduce problem  $Q$  to  $Q'$  (written  $Q \leq Q'$ ):
  - Given problem  $Q$
  - Know how to solve  $Q'$
  - Transform  $Q$  to  $Q'$
  - Solve  $Q'$
  - Deduce solution to  $Q$ .
- Example: " $ax + b = 0$ "  $\leq$  " $Ax^2 + Bx + C = 0$ ".
- Question: If  $Q \leq Q'$ , which one is the hardest?

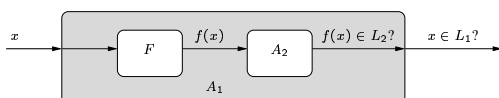
## Reduction of languages

- A language  $L_1$  is **polynomial-time reducible** to  $L_2$  (written  $L_1 \leq_P L_2$ ) if there exists a polynomial-time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that for all  $x \in \{0, 1\}^*$ :

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

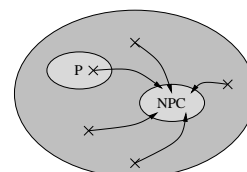
- **Lemma 36.3**: If  $L_1 \leq_P L_2$  then  $L_2 \in \text{P}$  implies  $L_1 \in \text{P}$ .

**Proof**: Construct a poly-time algorithm  $A_1$  deciding  $L_1$  as:



## Definition of NP-complete

- If  $L_1 \leq_P L_2$  then " $L_2$  is at least as hard as  $L_1$ ."
- NP-complete problems: the hardest problems in NP
- $L \subseteq \{0, 1\}^*$  is **NP-complete** if
  1.  $L \in \text{NP}$
  2.  $\forall L' \in \text{NP} : L' \leq_P L$  (NP-hard).



## Properties of NPC

- $P = NP$  ? closely related to the class of NP-complete problems:
- **Theorem 36.4:**
  1. If any NP-complete problem is polynomial-time solvable, then  $P = NP$ .  
**Proof:** Suppose  $L \in P$  and  $L \in NPC$ . For any  $L' \in NP$ ,  $L' \leq_P L$ . Thus,  $L' \in P$ .
  2. If any problem in NP is *not* polynomial-time solvable, then all NP-complete problems are not polynomial-time solvable.  
**Proof:** Assume  $L \in NP$  and  $L \notin P$ . Let  $L' \in NPC$  and assume  $L' \in P$ . But since  $L \leq_P L'$ , it follows that  $L \in P$ .

## Proving $L \in NPC$ ?

- To prove  $L \in NPC$ :
  1. Prove that  $L \in NP$
  2.  $\forall L' \in NP$ : prove that  $L' \leq_P L$ .
- If we know that  $L' \in NPC$ , step 2 can be replaced with
  - 2'. Prove that  $L' \leq_P L$ .
- We need that first NP-complete problem !

## CIRCUIT-SAT $\in$ NPC

$CIRCUIT-SAT = \{ \langle C \rangle : C \text{ is a satisfiable combinational circuit} \}$ .

- **Theorem:** CIRCUIT-SAT is NP-complete

**Proof:**

1. CIRCUIT-SAT  $\in$  NP.
2. CIRCUIT-SAT is NP-hard.

## Proof that CIRCUIT-SAT is NP-hard

- Prove:  $\forall L' \in NP : L' \leq_P L$ .
- Idea: "Take any problem in NP and show how to phrase it as a CIRCUIT-SAT problem."
- Formally: Let  $L \in NP$ . Construct poly-time  $F$  computing reduction  $f$ :  
$$\forall x \in \{0, 1\}^* : f(x) \text{ is a circuit } C \text{ s.t. } x \in L \Leftrightarrow C \in CIRCUIT-SAT.$$
- Since  $L \in NP$ , there exists an algorithm  $A$  that verifies  $L$  in poly-time  $T(n)$ .

## Proof that CIRCUIT-SAT is NP-hard (2)

- $F$  constructs circuit  $C$  as  $T(n)$  copies of  $M$ .
- Key:  $C$  is satisfiable if and only if  $\exists y : A(x, y) = 1$ .
- Remains: prove  $F$  runs in poly-time,  $C$  has poly-size and lots of other details...
- *We got the first NP-complete problem!*

## Proving that $L$ is NP-complete

- **Lemma 36.8:** If  $L' \leq_P L$  for some  $L' \in NPC$ , then  $L$  is NP-hard.  
**Proof:** Since  $L' \in NPC$ , for all  $L'' \in NP : L'' \leq_P L'$ . By transitivity,  $L'' \leq_P L$ .
- Proving that a problem  $L$  is NP-complete:
  1. Prove  $L \in NP$ .
  2. Select a known NP-complete language  $L'$ .
  3. Describe algorithm that computes  $f : L' \rightarrow L$ .
  4. Prove that  $f$  satisfies  $x \in L'$  if and only if  $f(x) \in L$ .
  5. Prove that the algorithm computing  $f$  runs in polynomial time.

## SAT

- Given a Boolean formula  $\Phi$  composed of

- Boolean variables:  $x_1, x_2, x_3, \dots$
- Boolean connectives:  $\wedge, \vee, \neg, \rightarrow, \dots$
- Parenthesis

$$\text{SAT} = \{ \langle \Phi \rangle : \Phi \text{ is a satisfiable Boolean formula} \}$$

- Example:

$$\Phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2.$$

is satisfiable ( $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$ ), thus  $\Phi \in \text{SAT}$ .

## SAT is NP-complete

- Theorem:** SAT is NP-complete.

**Prove:**

- Prove  $\text{SAT} \in \text{NP}$ .
- Select a known NP-complete language  $L' = \text{CIRCUIT-SAT}$ .
- Describe algorithm that computes  $f : \text{CIRCUIT-SAT} \rightarrow \text{SAT}$ .
- Prove that  $f$  satisfies  $x \in \text{CIRCUIT-SAT}$  if and only if  $f(x) \in \text{SAT}$ .
- Prove that the algorithm computing  $f$  runs in polynomial time.

## SAT is NP-complete (2)

- $f$ ? Given a circuit, how do we construct a Boolean formula such that the circuit is satisfiable if and only if the Boolean formula is satisfiable?

- Introduce a new variable for each wire in circuit  $C$ .

- Example:  $x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)$ .

- Construct  $\Phi$  as

$$\begin{aligned} \Phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \\ & \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \\ & \wedge \dots \end{aligned}$$

## SAT is NP-complete (3)

- Given  $C$ , it is straightforward to produce  $\Phi$  in poly-time.

- Show that:

$$C \text{ is satisfiable} \Leftrightarrow \Phi \text{ has a satisfying assignment.}$$

- Then

$$\text{CIRCUIT-SAT} \leq_P \text{SAT}.$$

- And we have shown that SAT is NP-complete.

## 3-CNF-SAT

- A Boolean formula in **3-conjunctive normal form**:

$$\Phi = (x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

- 3-CNF-SAT: is a Boolean formula in 3-conjunctive normal form satisfiable?

- Theorem:** 3-CNF-SAT is NP-complete

**Proof:**

- 3-CNF-SAT  $\in$  NP.
- 3-CNF-SAT is NP-hard.

## 3-CNF-SAT is NP-hard

- We show that  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .

- Transform  $\Phi$  to 3-CNF.

- Step 1: Construct binary parse tree for  $\Phi$  and express  $\Phi$  as

$$\begin{aligned} \Phi' = & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_3)) \\ & \wedge \dots \end{aligned}$$

- Note:  $\Phi'$  is a conjunctions of clauses, each of which has at most 3 literals.

### 3-CNF-SAT is NP-hard (2)

- Step 2: Transform  $\Phi'_i$  into conjunctive normal form using a truth-table for each operator.
- For  $\Phi'_1 = (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$  the truth-table is:

$y_1$	$y_2$	$x_2$	$y_1 \leftrightarrow (y_2 \wedge \neg x_2)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

### 3-CNF-SAT is NP-hard (3)

- Step-2 (cont.):

$$\neg\Phi'_1 = (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2)$$

- Applying DeMorgan's laws:

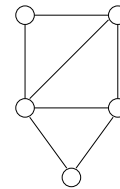
$$\Phi''_1 = (y_1 \vee \neg y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_2)$$

### 3-CNF-SAT is NP-hard (4)

- Step-3: Make sure each clause has *exactly* three literals.
- Add two new variables  $p$  and  $q$ .
  - If  $C_i = (l_1 \vee l_2)$ : Include  $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$  in  $\Phi'''$ .
  - If  $C_i = (l)$ : Include  $(l \vee p \vee q) \wedge (l \vee \neg p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee \neg q)$  in  $\Phi'''$ .
- Key observation  $\Phi$  is satisfiable if and only if  $\Phi'''$  is.
- Transformation takes polynomial time.

### Other NP-complete problems: Clique

- Given an undirected graph  $G = (V, E)$ .
- A **clique** is a subset  $V' \subseteq V$  such that each pair of vertices in  $V'$  is connected by an edge in  $E$ .



- The clique problem is

$$\text{CLIQUE} = \{ \langle G, k \rangle : G \text{ is a graph with a clique of size } k \}.$$

### Clique is NP-complete

- **Theorem:** CLIQUE is NP-complete
- 3-CNF-SAT  $\leq_P$  CLIQUE.
- That is, given  $\Phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$  on 3-CNF, construct a graph  $G$  such that  $\Phi$  is satisfiable if and only if  $G$  has a clique of size  $k$ .
- For  $C_r = (l_1^r \vee l_2^r \vee l_3^r)$  in  $\Phi$ , add vertices  $v_1^r, v_2^r, v_3^r$  in  $V$ .
- Add edge  $(v_i^r, v_j^s)$  to  $E$  if
  1.  $r \neq s$ , and
  2.  $l_i^r$  is not the negation of  $l_j^s$ .

### Correctness of reduction

- $\Phi$  is satisfiable  $\Rightarrow G$  has a clique of size  $k$ .

**Proof:** At least one literal in each clause is **true**. Let  $V'$  be a set of vertices corresponding to selecting a **true** literal in each clause.

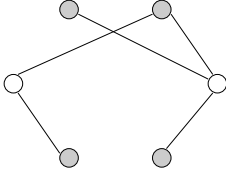
Claim:  $V'$  is a clique of size  $k$ . Pick  $v_i^r$  and  $v_j^s$  in  $V'$ . Since both are true, they cannot be each other negation and  $(v_i^r, v_j^s)$  is in  $E$ .

- $G$  has a clique  $V'$  of size  $k \Rightarrow \Phi$  is satisfiable.

**Proof:** There are no edges between vertices in the same triple. Assign **true** for  $l_i^r \in V'$  (we wont assign **true** to both  $x_i$  and  $\neg x_i$  due to the construction of  $G$ ). Each clause is satisfied, thus  $\Phi$  is satisfied.

## More NP-complete problems: Vertex Cover

- Given an undirected graph  $G = (V, E)$ .
- A subset  $V' \subseteq V$  is a **cover** of  $G$  if for  $(u, v) \in E$ , either  $u \in V'$  or  $v \in V'$ .

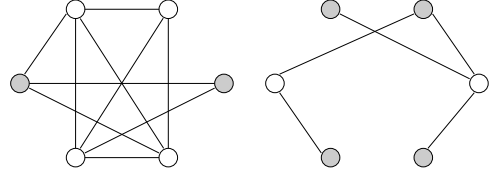


- The vertex cover is

VERTEX-COVER =  $\{\langle G, k \rangle : \text{graph } G \text{ has a vertex-cover of size } k\}$ .

## Vertex Cover is NP-complete

- Theorem:** Vertex cover is NP-complete
- CLIQUE  $\leq_P$  VERTEX-COVER.
- Given  $G = (V, E)$ , construct the *complement*  $\bar{G} = (V, \bar{E})$ , where  $\bar{E} = \{(u, v) : (u, v) \notin E\}$ .



## Vertex Cover is NP-complete (2)

- $G$  has a clique of size  $k \Rightarrow \bar{G}$  has a vertex cover of size  $|V| - k$ .

Let  $V' \subseteq V$  be a clique of  $G$  with  $|V'| = k$ . Claim:  $V - V'$  is a vertex cover of  $\bar{G}$ . Let  $(u, v)$  be any edge in  $\bar{E}$ , then  $(u, v) \notin E$  and either  $u$  or  $v$  is not in  $V'$  (since  $V'$  is a clique). Thus, either  $u$  or  $v$  is in  $V - V'$  and edge  $(u, v)$  is covered by  $V - V'$ .

- $\bar{G}$  has a vertex cover  $V' \subseteq V$  where  $|V'| = |V| - k \Rightarrow G$  has a clique of size  $k$ .

For all  $u, v \in V : (u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'$ . Contrapositive: For all  $u, v \in V : (u, v) \in E \Rightarrow u \notin V' \text{ and } v \notin V'$ . Thus,  $V - V'$  is a clique.

## More NP-complete problems: Subset-sum

- Given a finite set  $S \subset \mathbf{N}$  and  $t \in \mathbf{N}$ .
- Is there a subset  $S' \subseteq S$  whose elements sum to  $t$ ?
- For example, if

$$S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$$

and  $t = 3754$  then the subset

$$S' = \{1, 16, 64, 256, 1040, 1093, 1284\}$$

is a solution.

- Formally,

$$\text{SUBSET-SUM} = \{\langle S, t \rangle : \text{there exists a subset } S' \subseteq S \text{ such that } t = \sum_{s \in S'} s\}.$$