CS244: Theory of Computation

Fu Song
ShanghaiTech University

Fall 2022

- Even when a problem is decidable, and thus computationally solvable in principle
- it might not be solvable in practice, since the optimal Turing machine which decides this problem could require astronomical time or memory

Part Three. Complexity Theory

An investigation of the time, memory, or other resources required for solving computational problems

Outline

Time Complexity

Outline

$A = \{0^k 1^k \mid k \geq 0\}$

M_1 on w:

```
1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape.
3.     Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still
   remain after all the 0s have been crossed off, reject.
   Otherwise if neither 0s nor 1s remain on the tape, accept.
```

Time complexity?

Time complexity of M_1

1. Analyze the running time of M_1 on every $x \in \Sigma^*$:

$$f_1 : \Sigma^* \to \mathbb{N}.$$

2. Analyze the worse-case running time of M_1 on inputs of length $n \in \mathbb{N}$, $f_2 : \mathbb{N} \to \mathbb{N}$. In particular,

$$f_2(n) = \max_{x \in \Sigma^n} f_1(x).$$

3. Analyze the average-case running time of M_1 on inputs of length $n \in \mathbb{N}$, $f_3 : \mathbb{N} \to \mathbb{N}$. In particular,

$$f_3(n) = \frac{\sum_{x \in \Sigma^n} f_1(x)}{|\Sigma|^n}.$$

Worse-case analysis

Definition

Let M be a deterministic Turing machine that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

If $f(n)$ is the running time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine.

Customarily we use n to represent the length of the input.

Big-O notation

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be two functions. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c \cdot g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an upper bound for $f(n)$, or more precisely, that $g(n)$ is an asymptotic upper bound for $f(n)$, to emphasize that we are suppressing constant factors.

$\mathbb{R}^+$ denotes non-negative real numbers

Examples

1. $5n^3 + 2n^2 + 22n + 6 = O(n^3)$.
2. Let $b \geq 2$. Then
$$\log_b n = \frac{\log_2 n}{\log_2 b}$$
   Hence, $\log_b(n) = O(\log n)$.
3. $3n \log_2 n + 5n \log_2 \log_2 n + 2 = O(n \log n)$.
4. $2^{10n^2+7n-6} = 2^{O(n^2)}$.

n^c for $c > 0$ is a polynomial bound.

$2^{\left(n^\delta\right)}$ for $\delta > 0$ is an exponential bound.

Small o-notation

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be two functions. Say that $f(n) = o(g(n))$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

In other words, $f(n) = o(g(n))$ means that for any real number $c > 0$, a number n_0 exists, where $f(n) < c \cdot g(n)$ for all $n \geq n_0$.

Note: $f(n) = O(g(n))$ means that positive integer number $c > 0$ and n_0 exists, where $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Examples

1. $\sqrt{n} = o(n)$.
2. $n = o(n \log \log n)$.
3. $n \log \log n = o(n \log n)$.
4. $n \log n = o(n^2)$.
5. $n^2 = o(n^3)$.

$A = \{0^k 1^k \mid k \geq 0\}$

M_1 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, reject. Otherwise if neither 0s nor 1s remain on the tape, accept.

Time analysis:

- First stage scans the tape to verify the input is of the form 0^*1^*, taking n steps. Then the machine repositions the head at the left-hand end of the tape, again using n steps. In total $2n = O(n)$ steps.
- In stages 2 and 3, the machine repeatedly scans the tape and crosses off a 0 and 1 on each scan. Each scan uses $O(n)$ steps. Because each scan crosses off two symbols, at most $n/2$ scans can occur. So the total time taken by stages 2 and 3 is $(n/2)O(n) = O(n^2)$.
- In stage 4, the machine makes a single scan to decide whether to accept or reject, hence requires time $O(n)$.

The overall running time

$$O(n) + O(n^2) + O(n) = O(n^2).$$

Time classes

Definition

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. Define the time complexity class

$$\mathsf{TIME}\big(t(n)\big),$$

to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Example

$\{0^k 1^k \mid k \geq 0\} \in \mathsf{TIME}\big(n^2\big)$.

A better algorithm

M_2 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and 1s remain on the tape.
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, then reject.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, then accept. Otherwise, reject.

Time analysis

M_2 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and 1s remain on the tape.
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, then reject.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, then accept. Otherwise, reject.

1. Every stage takes $O(n)$ time.
2. Stages 1 and 5 are executed once, hence total $O(n)$ time.
3. Stage 4 crosses of at least half of the 0s and 1s each time it is executed, hence at most $1 + \log_2 n$ iterations.

   Thus the total time of stages 2, 3, and 4 is $(1 + \log_2 n)O(n) = O(n \log n)$.

The overall running time of M_2 is

$$O(n) + O(n \log n) = O(n \log n).$$

Can we do even better than $O(n \log n)$?

Theorem

Every language that can be decided in $o(n \log n)$ time on a single-tape Turing machine is regular. [1]

Quite strange: regular language can be decided in $\mathrm{TIME}(n)$, there is not any language in $\mathrm{TIME}(o(n \log n)) \setminus \mathrm{TIME}(O(n))$.

[1] K. Kobayashi. On the structure of one-tape nondeterministic Turing machine time hierarchy. Theor. Comput. Sci., 40 (1985), 175-193.

$\{0^k1^k \mid k \geq 0\}$ in linear time on a 2-tape TM

M_3 on w:

1. Scan across tape 1 and reject if a 0 is found to the right of a 1.
2. Scan across the 0s on tape 1 until the first 1. At the same time copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, then reject.
4. If all the 0s have now been crossed off, then accept. if any 0s remain, then reject.
5. If no 0s and no 1s remain on the tape, then accept. Otherwise, reject.

Complexity relationships among models

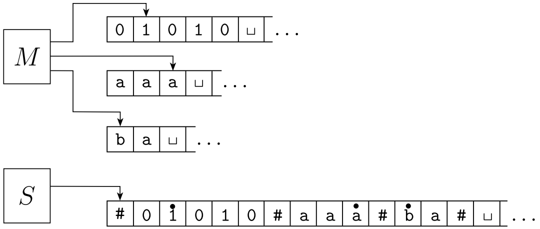
Theorem

Let $t(n)$ be a function with $t(n) \geq n$. The every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Proof (1)

We simulate an M with k tapes by a single-tape S.

- S uses # to separate the contents of the different tapes.
- S keeps track of the locations of the heads by writing a tape symbol with a dot above it to mark the place where the head on that tape would be.



Proof (2)

On input $w = w_1 \cdots w_n$:

1. First S puts its tape into the format that represents all k tapes of M:

   $$\# \dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#.$$

   Time: $O(n) = O(t(n))$.
2. To determine the symbols under the virtual heads, S scans its tape from the first $\#$, which marks the left-hand end, to the $(k+1)$st $\#$, which marks the right-hand end. Time: $O(t(n))$.
3. Then S makes a second pass to update the tapes according to the way that Ms transition function dictates.

   If S moves one of the virtual heads to the right onto a $\#$, then S writes $\sqcup$ on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the right.

   Time: $O(k \cdot t(n)) = O(t(n))$.
4. Go back to 2.

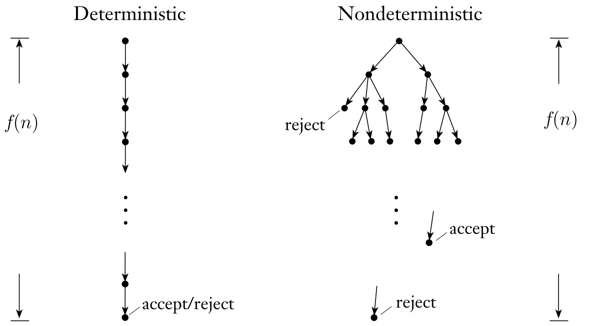
In total: $O(t^2(n))$

Nondeterministic Turing machines

Definition

Let N be a nondeterministic Turing machine that is a decider. The running time of N is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n.

Measuring deterministic and nondeterministic time



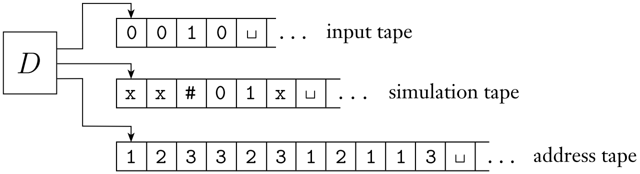
Theorem

Let $t(n)$ be a function with $t(n) \geq n$, then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Proof (1)

We simulate a nondeterministic TM N by a deterministic TM D.

1. D try all possible branches of N's nondeterministic computation.
2. If D ever finds the accept state on one of these branches, it accepts.


D
0 0 1 0 ␣ . . . input tape
x x # 0 1 x ␣ . . . simulation tape
1 2 3 3 2 3 1 2 1 1 3 ␣ . . . address tape

Proof (2)

- On an input of length n, every branch of N's nondeterministic computation tree has a length of at most $t(n)$.

  Every node in the tree can have at most b children, where b is the maximum number of legal choices given by N's transition function.

  Thus, the total number of leaves in the tree is at most $b^{t(n)}$.
- The total number of the nodes in that tree is less than twice the maximum number of leaves, hence, $O(b^{t(n)})$. The time it takes to start from the root and travel down to a node is $O(t(n))$. Hence the total running time of D is $O(t(n)b^{t(n)}) = 2^{O(t(n))}$.
- D has 3 tapes, thus can be simulated by a single-tape TM in time

$$\left(2^{O(t(n))}\right)^2 = 2^{O(t(n))}.$$

Outline

The Class **P**

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}\left(n^k\right).$$

1. **P** is invariant for all models of computation that are polynomially equivalent to the deterministic single-tap machine.
2. **P** roughly corresponds to the class of problems that are realistically solvable on a computer.

Examples of problems in **P**

Reasonable encodings

- We continue to use $\langle \cdot \rangle$ to indicate a reasonable encoding of one or more objects into a string.
- Unary encoding of n as $\underbrace{11 \cdots 11}_{n \text{ times}}$ is exponentially larger than the standard binary encoding of n, hence not reasonable.
- A graphs can be encoded either by listing its nodes and edges, i.e., its adjacency list, or it adjacency matrix, where the (i, j)th entry is 1 if there is an edge from node i to node j and 0 if not.

The path problem

$$PATH = \Big\{ \langle G, s, t \rangle \;\Big|\; G \text{ is a directed graph}$$
$$\text{that has a directed path from } s \text{ and } t \Big\}.$$

Theorem
PATH $\in$ **P**.

Brute-force search does not work, there may be roughly m^m paths, where m denotes the number of nodes

M on input $\langle G, s, t\rangle$: (breadth-first search)

```
1. Place a mark on node $s$.
2. Repeat the following until no additional nodes are marked:
3.     Scan all the edges of $G$. If an edge $(a, b)$ is found going from a
       marked node $a$ to an unmarked node $b$, mark node $b$.
4. If $t$ is marked, accept. Otherwise, reject
```

Testing relatively prime

$$\mathit{RELPRIME} = \big\{\langle x, y\rangle \;\big|\; x \text{ and } y \text{ are relatively prime, i.e., } 1 \text{ is the greatest common divisor of } x \text{ and } y\big\}.$$

Theorem
$\mathit{RELPRIME} \in \mathbf{P}$.

Brute-force search all the possible divisors does not work, exponential number of potential divisors

The Euclidean algorithm

Recall the greatest common divisor $\gcd(x, y)$ is the largest integer that divides both x and y.

```
$E$ on $\langle x, y\rangle$: $\gcd(x, y) = \gcd(y, x \pmod{y})$
  1. Repeat until $y = 0$:
  2.     Assign $x \leftarrow x \pmod{y}$.
  3.     Exchange $x$ and $y$.
  4. Output $x$.
$R$ on $\langle x, y\rangle$:
  1. Run $E$ on $\langle x, y\rangle$.
  2. If the result is 1, then accept. Otherwise, reject.
```

Time analysis

E on $\langle x, y \rangle$: $\gcd(x,y) = \gcd(y, x \pmod{y})$

```
1. Repeat until $y = 0$:
2.     Assign $x \leftarrow x \pmod{y}$.
3.     Exchange $x$ and $y$.
4. Output $x$.
```

We show that E runs in polynomial time.

- Every execution of stage 2 with $y \leq x$ cuts the value x at least by half (except possibly the first iteration).
    - If $\frac{x}{2} \geq y$, then $x \pmod{y} < y \leq \frac{x}{2}$
    - If $\frac{x}{2} < y$, then $x \pmod{y} = x - y < \frac{x}{2}$
- Thus, the maximum number of times that stages 2 and 3 are executed is the lesser of $2\log_2 x$ and $2\log_2 y$.

Testing context-freeness

Theorem

Every context-free language is a member of **P**.

Recall (1)

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \to BC \quad \text{and} \quad A \to a$$

where a is a terminal, and A, B, and C are variables – except that B and C may be not the start variable. In addition, we permit the rule $S \to \varepsilon$, where S is the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Theorem

Let G be CFG in Chomsky normal form, and G generates w with $w \neq \varepsilon$. Then any derivation of w has $2|w| - 1$ steps.

Recall (2)

S on $\langle G, w\rangle$:

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2|w| - 1$ steps; except if $|w| = 0$, then instead check whether there is a rule $S \to \varepsilon$.
3. If any of these derivations generates w, then accept; otherwise reject.

The running time of S is $2^{O(n)}$.

Dynamic programming

Let $w = w_1 \cdots w_n$ be an input string.

For every $1 \leq i \leq j \leq n$ we will compute

$$\mathit{table}(i,j) = \text{the collection of non-terminals}$$
$$\text{that can generate the substring } w_i w_{i+1} \ldots w_j.$$

Dynamic programming (cont'd)

```
$D$ on $w = w_1 \cdots w_n$:
 1. For $w = \varepsilon$, if $S \to \varepsilon$ is a rule, then accept; else reject.
 2. For $i = 1$ to $n$:
 3.     For each non-terminal $A$:
 4.         Test whether $A \to b$ is a rule, where $b = w_i$.
 5.         If so, place $A$ in $table(i, i)$.                    [$|N| \times n$ times]
 6. For $\ell = 2$ to $n$:                      [length of the substring $w_i \cdots w_j$]
 7.     For $i = 1$ to $n - \ell + 1$:
 8.         Let $j = i + \ell - 1$
 9.         For $k = i$ to $j - 1$:
10.             For each rule $A \to BC$:
11.             If $B \in table(i, k)$ and $C \in table(k + 1, j)$, then put $A$ in
    $table(i, j)$.                                                   [$n^3$ times]
12. If $S \in table(1, n)$, then accept; else reject.
```

Outline

The Class **NP**

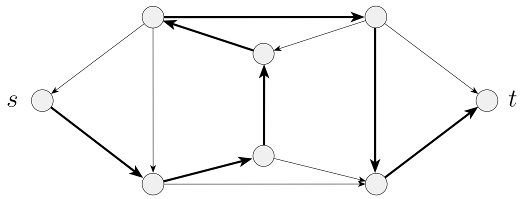
Hamiltonian path

Definition

A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once.

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ and } t\}.$$

Hamiltonian path (cont'd)



Polynomial verifiability

Even though we don't know how to determine fast whether a graph contains a Hamiltonian path, if such a path were discovered somehow (perhaps using the exponential time algorithm), we could easily convince someone else of its existence simply by presenting it.

In other words, verifying the existence of a Hamiltonian path may be much easier than determining its existence.

Testing composite

Definition

A natural number is composite if it is the product of two integers > 1.

$$\mathit{COMPOSITES} = \left\{ x \;\middle|\; x = pq \text{ for integers } p, q > 1 \right\}.$$

Verifiers

Definition

A verifier for a language A is an algorithm V, where

$$A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}.$$

We measure the time of a verifier only in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w.

A language A is polynomial verifiable if it has a polynomial time verifier.

The string c in the above definition is a certificate, or proof, of membership in A. For polynomial verifiers, the certificate has polynomial length (in the length of w).

Certificates

For *HAMPATH*, a certificate for $\langle G, s, t\rangle \in HAMPATH$ is a Hamiltonian path from s to t.

For *COMPOSITES*, a certificate for x is one of its divisors.

The class **NP**

Definition

NP is the class of languages that have polynomial time verifiers.

Nondeterministic polynomial Turing machines

Theorem

A language is in **NP** *if and only if it is decided by some nondeterministic polynomial time Turing machines.*

Proof (1)

Theorem

A language is in **NP** *if and only if it is decided by some nondeterministic polynomial time Turing machines.*

($\Rightarrow$) Assume a language is in **NP**. Let V be the verifier V that is a n^k time TM.

N on w with $n = |w|$:

1. Nondeterministically select string c of length at most n^k.
2. Run V on $\langle w, c \rangle$.
3. If V accepts, then accept; otherwise reject.

Proof (2)

Theorem

A language is in **NP** *if and only if it is decided by some nondeterministic polynomial time Turing machines.*

($\Leftarrow$) Assume that a language A is decided by a polynomial time NTM N.

V on $\langle w, c \rangle$:

1. Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step.
2. If this branch of N's computation accepts, then accept; otherwise reject.

Nondeterministic time complexity classes

Definition

$$\mathsf{NTIME}(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}.$$

Corollary

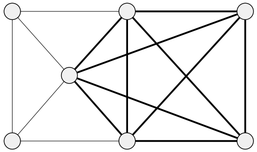
$$\mathbf{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}\left(n^k\right).$$

Examples of problems in **NP**

The clique problem

Definition

A clique in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A *k*-clique is a clique that contains *k* nodes.



A graph with a 5-clique.

The clique problem (cont'd)

$$\mathit{CLIQUE} = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}.$$

Theorem

CLIQUE is in **NP**.

Proof (1)

Construct a polynomial-time verifier V:

V on $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, then accept; otherwise reject.

Proof (2)

Alternative proof: construct a polynomial-time NTM decider N

N on $\langle G, k\rangle$:

1. Nondeterministically select a subset c of k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If yes, then accept; otherwise reject.

The subset-sum problem

$$SUBSET\text{-}SUM = \Big\{ \langle S, t \rangle \;\Big|\; S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_\ell\} \subseteq S, \text{ we have } \sum_{i \in [\ell]} y_i = t \Big\}.$$

Theorem

SUBSET-SUM is in **NP**.

Two Proofs

Theorem
SUBSET-SUM is in **NP**.

V on input $\langle\langle S, t\rangle, c\rangle$: (polynomial-time verifier)

1. Test whether c is a collection of numbers that sum to t.
2. Test whether S contains all the numbers in c.
3. If both pass, accept; otherwise, reject

N on input $\langle S, t\rangle$: (polynomial-time NTM)

1. Nondeterministically select a subset c of the numbers in S.
2. Test whether c is a collection of numbers that sum to t.
3. If the test passes, accept; otherwise, reject

Quiz: Hamiltonian path

Definition
A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once.

$$\mathit{HAMPATH} = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ and } t\}.$$

Theorem
HAMPATH is in **NP**.

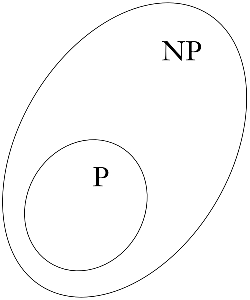
The **P** versus **NP** question

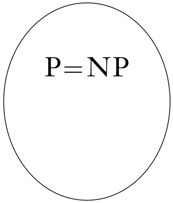
P = the class of languages for which membership can be decided quickly.

NP = the class of languages for which membership can be verified quickly.

Two possibilities

Open Problem!!!





Outline

The **NP**-Completeness

In 1970s, Stephen Cook and Leonid Levin discovered certain problems in **NP** whose individual complexity is related to that of the entire class.

If a polynomial time algorithm exists for any of these problems, all problems in **NP** would be polynomial time solvable.

These problems are called **NP**-complete.

Satisfiability problem

- Boolean variables are assigned to TRUE (1) or FALSE (0).
- Boolean operations are AND, OR, and NOT.
- A Boolean formula is an expression involving Boolean variables and operations.
- A Boolean formula is satisfiable is some assignment makes the formula evaluate to 1.
- The satisfiability problem is to test whether a Boolean formula is satisfiable, i.e.,

$$\text{SAT} = \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable Boolean formula}\}.$$

Theorem

SAT $\in$ $\mathbf{P}$ *if and only if* $\mathbf{P} = \mathbf{NP}$.

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some polynomial time Turing machine exists that halts with just $f(w)$ on its tape, when started on any input w.

Definition

Let $A, B \subseteq \Sigma^*$. Then A is polynomial time mapping reducible, or simply polynomial time reducible, to B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w

$$w \in A \iff f(w) \in B.$$

The function f is called the polynomial time reduction of A to B.

Theorem
If $A \leq_{\mathrm{P}} B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.

N on input w:

1. Compute $f(w)$ which reduces A to B.
2. Run the decider M of B on input $f(w)$ and output whatever M outputs.

3SAT

- A literal is a Boolean variable or a negated Boolean variable.
- A clause is several literals connected with $\vee$s.
- A Boolean formula is in conjunctive normal form, called a cnf-formula, if it comprises several clauses connected with $\wedge$s.
- A Boolean formula is in disjunctive normal form, called a dnf-formula, if it comprises several $\wedge$-connected clauses connected with $\vee$s.
- A Boolean formula is a 3cnf-formula if all the clauses have three literals.
- Let
$$\text{3SAT} = \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable 3cnf-formula}\}.$$

Boolean formula to CNF

Theorem

Any Boolean formula Φ can be translated into an equivalent CNF formula Ψ whose size is only polynomial in the size of ϕ.

Use De Morgan's law and the distributive property to convert it to CNF?

$$\bigvee_{i=1}^{n}(a_i \wedge b_i) \equiv \bigwedge_{i=1}^{n} \bigvee_{x_i \in \{a_i, b_i\}} x_i$$

Tseytin transformation for Φ

1. For each formula φ of $p \wedge q$, $p \vee q$ or $\neg q$, we introduce a fresh variable x_φ and a fresh clause $x_\varphi \leftrightarrow \varphi$
2. For each formula φ of the form $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ or $\neg\varphi_1$, we introduce a fresh variable x_φ and a fresh clause $x_\varphi \leftrightarrow x_{\varphi_1} \wedge x_{\varphi_2}$, $x_\varphi \leftrightarrow x_{\varphi_1} \vee x_{\varphi_2}$ or $x_\varphi \leftrightarrow \neg x_{\varphi_1}$
3. Conjunct all the fresh clauses and the fresh variable for Φ together, which can be transformed into CNF directly, resulting in Ψ

Boolean formula to CNF

Tseytin transformation for $\Phi = ((p \vee q) \wedge r) \vee (\neg s)$

1. $x_1 \leftrightarrow (p \vee q)$
   $\equiv (x_1 \rightarrow (p \vee q)) \wedge ((p \vee q) \rightarrow x_1)$
   $\equiv (\neg x_1 \vee p \vee q)) \wedge (\neg(p \vee q) \vee x_1)$
   $\equiv (\neg x_1 \vee p \vee q) \wedge (\neg p \vee x_1) \wedge (\neg q \vee x_1) \equiv \psi_1$
2. $x_2 \leftrightarrow x_1 \wedge r$
   $\equiv (x_2 \rightarrow (x_1 \wedge r)) \wedge ((x_1 \wedge r) \rightarrow x_2)$
   $\equiv (\neg x_2 \vee (x_1 \wedge r)) \wedge (\neg(x_1 \wedge r) \vee x_2)$
   $\equiv (\neg x_2 \vee x_1) \wedge (\neg x_2 \vee r) \wedge (\neg x_1 \vee \neg r \vee x_2) \equiv \psi_2$
3. $x_3 \leftrightarrow \neg s \equiv (x_3 \rightarrow \neg s) \wedge (\neg s \rightarrow x_3) \equiv (\neg x_3 \vee \neg s) \wedge (s \vee x_3) \equiv \psi_3$
4. $x_4 \leftrightarrow x_2 \vee x_3$
   $\equiv (x_4 \rightarrow (x_2 \vee x_3)) \wedge ((x_2 \vee x_3) \rightarrow x_4)$
   $\equiv (\neg x_4 \vee x_2 \vee x_3) \wedge (\neg(x_2 \vee x_3) \vee x_4)$
   $\equiv (\neg x_4 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_3 \vee x_4) \equiv \psi_4$

$\Psi \equiv x_4 \wedge \psi_1 \wedge \psi_2 \wedge \psi_3 \wedge \psi_4$ and $|\Psi| = c|\Phi|$

Boolean formula to DNF

Theorem

Any Boolean formula Φ can be translated into an equivalent DNF formula Ψ whose size is exponential in the size of ϕ.

Use De Morgan's law and the distributive property to convert it to DNF

$$\bigwedge_{i=1}^{n}(a_i \vee b_i) \equiv \bigvee_{i=1}^{n} \bigwedge_{x_i \in \{a_i, b_i\}} x_i$$

No better transformation is known

Theorem

3SAT *is polynomial time reducible to CLIQUE.*

Proof (1)

Let φ be a formula with k clauses such as

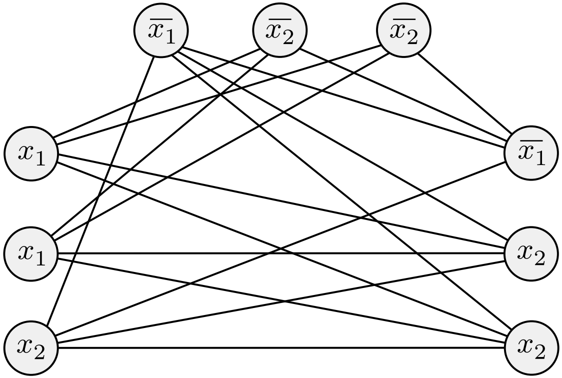
$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots (a_k \vee b_k \vee c_k).$$

The reduction generates a string $\langle G, k \rangle$.

1. The nodes in G are organized into k groups of three nodes $t_1, \ldots, t_k$. Each triple corresponds to one of the clauses, and each node in a triple corresponds to a literal in the associated clauses.
2. The edges of G connect all but two types of pairs of nodes in G.
   - No edge is present between nodes in the same triple.
   - no edge is present between two nodes with contradictory labels, e.g., x_2 and $\overline{x_2}$.

Proof (2)

$$(x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



Lemma

ψ is satisfiable iff G has a k-clique.

- Assume ψ is a satisfied by an assignment $f : X \to \{1, 0\}$.
- For each clause c_i, there is at least one literal l_{ij} is true under f, we add the corresponding node in the clique.
- The clique has k nodes, as we choose one node for each clause and there are k clauses
- All the nodes in the clique are connected with each other, as either x or $\overline{x}$ but not both in the clique and no pair of nodes come from the same clause

Lemma

ψ is satisfiable iff G has a k-clique.

- Assume G has a k-clique. Then, the k-clique cannot contains both x and $\overline{x}$ for each variable x.
- Let $f : X \to \{1, 0\}$ be an assignment such that $f(x) = 1$ iff a node labeled by x is in the clique.
- The k-clique cannot contains more than one node for each clause, as nodes of a clause are not connected.
- Then, for each clause, one corresponding node occurs in the k-clique.
- Then f satisfies ψ.

Definition

A language B is **NP**-complete if it satisfies two conditions:

1. B is in **NP**, and
2. every A in **NP** is polynomial time reducible to B.

Theorem

*If B is **NP**-complete and $B \in P$, then **P** = **NP**.*

All the **NP** problems can be reduced to B in polynomial time, then all the **NP** problems can be solved in **P**.

Theorem

*If B is **NP**-complete and $B \leq_{\mathrm{P}} C$ for C in **NP**, then C is **NP**-complete.*

All the **NP** problems can be reduced to B in polynomial time, and further can be reduced to C in polynomial time.

Theorem

SAT *is* **NP***-complete.*

Proof (1)

SAT is in **NP**, since a nondeterministic polynomial time Turing machine can

1. guess an assignment to a given formula φ,
2. accept if the assignment satisfies φ. (verifiable in polynomial-time)

How to prove that every **NP** problem is polynomial-time reducible to SAT?

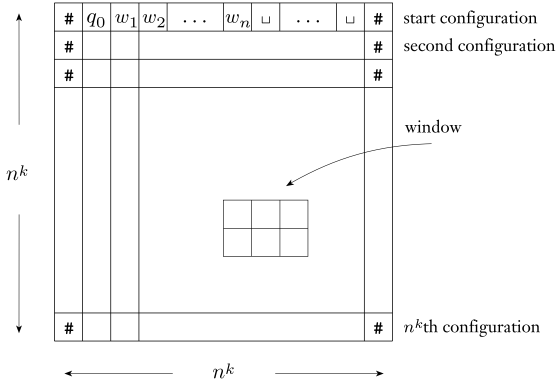
Theorem

A language is in **NP** *if and only if it is decided by some nondeterministic polynomial time Turing machines.*

Proof (2)

Let N be an NTM that decides a language A in time n^k for some $k \in \mathbb{N}$. We show $A \leq_P \text{SAT}$.

A tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations of the branch of the computation of N on input w.



Proof (3)

- We assume that each configuration starts and ends with a # symbol. Therefore, the first and last columns of a tableau are all #s.
- The first row of the tableau is the starting configuration of N on w, and each row follows the previous one according to N's transition function.
- A tableau is accepting if any row of the tableau is an accepting configuration.
- Every accepting tableau for N on w corresponds to an accepting computation branch of N on w. Thus, the problem of determining whether N accepts w is equivalent to the problem of determining whether an accepting tableau for N on w exists.

Proof (4)

On input w, the reduction produces a formula φ.

1. Let Q and Γ be the state set and tape alphabet of N. We set
$$C = Q \cup \Gamma \cup \{\#\}.$$
2. For each $i, j \in [n^k]$ and for each $s \in C$, we have a variable $x_{i,j,s}$.
3. Each of the $(n^k)^2$ entries of a tableau is called a cell.
4. If $x_{i,j,s}$ takes on the value 1, it means that the cell in row i and column j contains an s.

We represent the contents of the cells with the variables of φ.

Proof (5)

We design φ so that a satisfying assignment to the variables does correspond to an accepting tableau for N for w:

$$\varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{move}} \wedge \varphi_{\text{accept}}.$$

Proof (6)

$$\varphi_{\text{cell}} = \bigwedge_{i,j \in [n^k]} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right].$$

Each cell contains exactly one symbol from $C = Q \cup \Gamma \cup \{\#\}$

Proof (7)

$$\begin{aligned}\varphi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_1} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} .\end{aligned}$$

The first row is the starting configuration of N on w

Proof (8)

$$\varphi_{\mathrm{accept}} = \bigvee_{i,j \in [n^k]} x_{i,j,q_{\mathrm{accept}}}.$$

Some cell contains the accepting state, i.e., the tableau is accepting

Proof (9)

Finally, formula φ_{move} guarantees that each row of the tableau corresponds to a configuration that legally follows the preceding row's configuration according to N's rules.

It does so by ensuring that each 2×3 window of cells is legal.

We say that a 2×3 window is legal if that window does not violate the actions specified by N's transition function.

Proof (10)

Assume that:

- When in state q_1 with the head reading an a, N writes a b, stays in state q_1, and moves right, i.e., $\delta(q_1, a) = \{(q_1, b, R)\}$
- When in state q_1 with the head reading a b, N nondeterministically either
  1. writes a c, enters q_2, and moves to the left, or
  2. writes an a, enters q_2, and moves to the right.

  i.e., $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

Legal moves

Proof (11)

Assume that:

- When in state q_1 with the head reading an a, N writes a b, stays in state q_1, and moves right, i.e., $\delta(q_1, a) = \{(q_1, b, R)\}$
- When in state q_1 with the head reading a b, N nondeterministically either
  1. writes a c, enters q_2, and moves to the left, or
  2. writes an a, enters q_2, and moves to the right.

  i.e., $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

(a)

a	b	a
a	a	a

(b)

a	q_1	b
q_2	a	a

(c)

b	q_1	b
q_2	b	q_2

Illegal moves

Proof (12)

If the top row of the tableau is the start configuration and every window in the tableau is legal, each row of the tableau is a configuration that legally follows the preceding one.

Proof (13)

$$\varphi_{\mathrm{move}} = \bigwedge_{1 \le i,j < n^k} \text{the } (i,j)\text{-window is legal}.$$

We replace "the (i,j)-window is legal" by

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right).$$

Corollary

3SAT *is* **NP***-complete.*

Proof (1)

3SAT $\in$ **NP** is clear.

To show that every problem in **NP** can be reduced to 3SAT, we modify the previous reduction to SAT. Recall

$$\varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{move}} \wedge \varphi_{\text{accept}},$$

where

$$\varphi_{\text{cell}} = \bigwedge_{i,j \in [n^k]} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

$$\varphi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\llcorner\!\lrcorner} \wedge \ldots \wedge x_{1,n^k-1,\llcorner\!\lrcorner} \wedge x_{1,n^k,\#}$$

$$\varphi_{\text{move}} = \bigwedge_{1 \leq i,j < n^k} \bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge \ldots \wedge x_{i+1,j+1,a_6} \right)$$

$$\varphi_{\text{accept}} = \bigvee_{i,j \in [n^k]} x_{i,j,q_{\text{accept}}}.$$

Proof (2)

The formula is almost in conjunctive normal form, except

$$\varphi_{\mathrm{move}} = \bigwedge_{1 \le i,j < n^k} \bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge \ldots \wedge x_{i+1,j+1,a_6}\right).$$

Recall the distributive laws:

$$\left(a_{1,1} \wedge \ldots \wedge a_{1,n_1}\right) \vee \left(a_{2,1} \wedge \ldots \wedge a_{2,n_2}\right) = \bigwedge_{i \in [n_1], j \in [n_2]} a_{1,i} \vee a_{2,j}.$$

Therefore $\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge \ldots \wedge x_{i+1,j+1,a_6}\right)$ is equivalent to an cnf-formula of size at most

$$6^{|C|^6} = O(1),$$

where recall $C = Q \cup \Gamma \cup \{\#\}$.

Proof (3)

Now we need to convert the formula in cnf to one with three literals per clause:

1. In each clause that currently has one or two literals, we replicate one of the literals until the total number is three.
2. If a clause contains $\ell > 3$ literals

   $$(a_1 \vee a_2 \vee \ldots \vee a_\ell),$$

   we replace it with the $\ell - 2$ clauses

   $$(a_1 \vee a_2 \vee z_1) \wedge (\bar{z}_1 \vee a_3 \vee z_2) \wedge (\bar{z}_2 \vee a_4 \vee z_3) \wedge \ldots \wedge$$

   $$(\bar{z}_{\ell-4} \vee a_{\ell-2} \vee z_{\ell-3}) \wedge (\bar{z}_{\ell-3} \vee a_{\ell-1} \vee a_\ell).$$

Outline

Additional NP-complete problems

Theorem

3SAT *is polynomial time reducible to CLIQUE.*

Theorem

3SAT *is* **NP***-complete.*

Theorem

CLIQUE is in **NP***.*

Corollary

CLIQUE is **NP***-complete.*

Vertex cover

If G is an undirected graph, a vertex cover of G is a subset S of the nodes where every edge of G touches one of those nodes, formally, for every edge (x, y) in G, $x \in S \vee y \in S$.

$$\mathit{VERTEX\text{-}COVER} = \big\{\langle G, k\rangle \;\big|\; G \text{ is an undirected graph that has a } k\text{-vertex cover}\big\}.$$

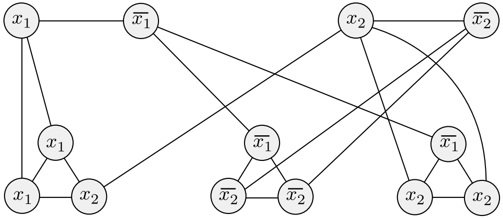
Theorem
VERTEX-COVER is **NP***-complete.*

- It is easy to prove that *VERTEX-COVER* is in **NP**.
- Hardness: reduce 3SAT to *VERTEX-COVER*.

3SAT to *VERTEX-COVER*

- For each variable $x \in X$ in ϕ, *VERTEX-COVER* contains two nodes (called variable nodes) labeled by x and $\overline{x}$ and an edge $(x, \overline{x})$. $x = 1$ corresponds to selecting the node labeled by x for the vertex cover
- For each clause $l_1 \vee l_2 \vee l_3 \in Cl$, *VERTEX-COVER* contains three nodes (called clause nodes) labeled by l_1, l_2 and l_3; and each of them are connected. Moveover, each pair of variable node and clause node is connected if they have same labels.
- The graph has $2|X| + 3|Cl|$ nodes.

$$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_2).$$



Proof

Lemma

ψ having ℓ clauses and m variables is satisfiable iff G has a $k = (m + 2\ell)$-vertex cover.

1. Assume ψ is satisfied by an assignment f.
2. For each variable, the variable node x is added into the cover iff $f(x) = 1$, the variable node $\overline{x}$ is added into the cover iff $f(x) = 0$. This adds m nodes into the cover.
3. For each clause $l_1 \vee l_2 \vee l_3$, we add two clause nodes (say labeled by l_1 and l_2) into the cover such that l_3 is true under f. This adds 2ℓ nodes into the cover.
4. We get $k = (m + 2\ell)$ nodes. This set must be a vertex cover.
   - Due to Item 2: edges between variable nodes are covered.
   - Due to Item 3: edges between clause nodes are covered.
   - The edges between variable nodes and clause nodes are also covered, as the clause node labeled by l_3 but not in the cover is true under f, then the variable node labeled by l_3 must be added into the cover

Proof

Lemma

ψ having ℓ clauses and m variables is satisfiable iff G has a $k = (m + 2\ell)$-vertex cover.

1. Assume G has a $k = (m + 2\ell)$-vertex cover.
   - In order to cover edges between variable nodes, for each variable x, the cover must contain a variable node labeled by either x or $\overline{x}$. Let f be the assignment such that $f(x) = 1$ iff the variable node labeled by x is in the cover.
   - In order to cover edges between clause nodes, the cover must contain two clause nodes of each clause.
2. The assignment f must satisfy ψ, as each edge ending with a clause node l_3 that is not in the cover must ends with a variable node l_3 from the cover.

The Hamiltonian path problem

The Hamiltonian path problem, i.e., *HAMPATH*, asks whether the input graph contains a path from s to t that goes through every node exactly once.

Theorem
HAMPATH is **NP***-complete.*

We have already proved that *HAMPATH* is in **NP**.

We show that 3SAT is polynomial-time reducible to *HAMPATH*.

How to encode truth of variables into *HAMPATH*?

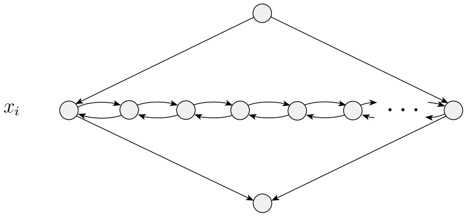
Proof (1)

We show 3SAT $\leq_P$ *HAMPATH*.

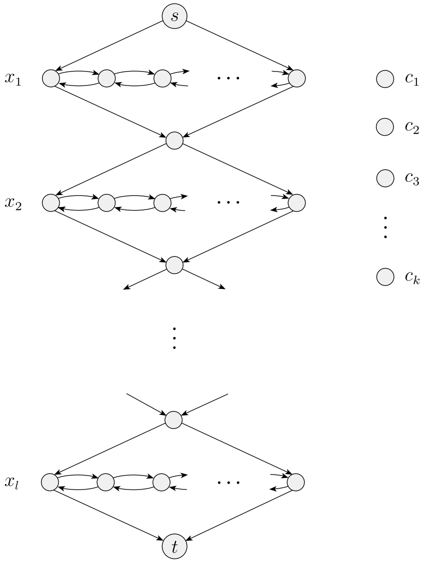
Let

$$\varphi = \left(l_1 \vee l_1' \vee l_1''\right) \wedge \left(l_2 \vee l_2' \vee l_2''\right) \wedge \ldots \wedge \left(l_k \vee l_k' \vee l_k''\right).$$

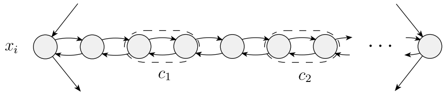
We represent each variable x_i with a diamond-shaped structure, and each clause as a single node.



Proof (2)

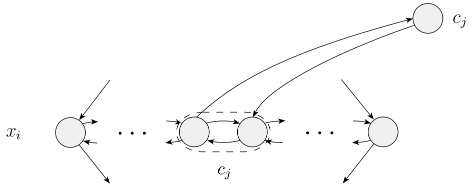


Proof (3)



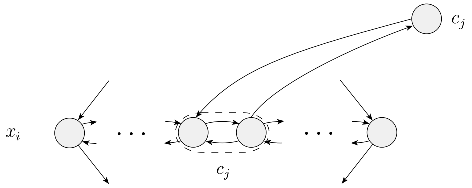
The horizontal nodes in a diamond structure

Proof (4)



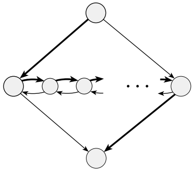
The additional edges when clause c_j contains x_i

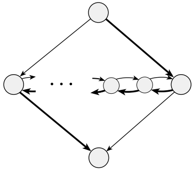
Proof (5)



The additional edges when clause c_j contains $\bar{x}_i$

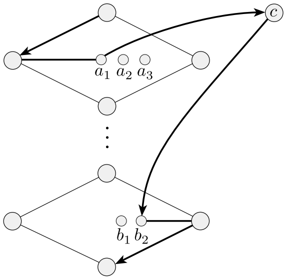
Proof (6)


zig-zag


zag-zig

- If x_i is assigned TRUE, the path zig-zags through the corresponding diamond.
- If x_i is assigned FALSE, the path zag-zigs.

Proof (7)



The above situation cannot occur.

The undirected Hamiltonian path problem

UHAMPATH, asks whether the undirected graph contains a path from s to t that goes through every node exactly once.

Theorem

UHAMPATH is **NP***-complete.*

It is easy to show that *UHAMPATH* is in **NP**. How to prove the another condition?

Proof

We show *HAMPATH* $\leq_{\mathrm{P}}$ *UHAMPATH*. Let G be a directed graph with nodes s and t.

1. Let u be a node in G with $s \neq u \neq t$. We replace it by a path of length 3
$$u^{\mathrm{in}} - u^{\mathrm{mid}} - u^{\mathrm{out}}.$$
2. s and t in G are replaced by $s^{\mathrm{out}} = s'$ and $t^{\mathrm{in}} = t'$.
3. If there is an edge from u to v in G, then in G' add an edge
$$u^{\mathrm{out}} - v^{\mathrm{in}}$$

It is easy to conclude

$$\langle G, s, t\rangle \in \mathit{HAMPATH} \iff \langle G', s', t'\rangle \in \mathit{UHAMPATH}.$$

The subset-sum problem

Recall

$$\mathit{SUBSET\text{-}SUM} = \Big\{ \langle S, t \rangle \;\Big|\; S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_\ell\} \subseteq S, \text{ we have } \sum_{i \in [\ell]} y_i = t \Big\}.$$

Theorem
SUBSET-SUM is **NP***-complete.*

We have already proved that *SUBSET-SUM* is in **NP**.

We show that 3SAT is polynomial-time reducible to *SUBSET-SUM*.

How to encode truth of variables into *SUBSET-SUM*?

Proof (1)

We show 3SAT $\leq_{\mathrm{P}}$ *SUBSET-SUM*. Let φ be a Boolean formula with variables $x_1, \ldots, x_\ell$ and clauses $c_1, \ldots, c_k$.

1. S consists of the numbers $y_1, z_1, \ldots, y_\ell, z_\ell$, and $g_1, h_1, \ldots, g_k, h_k$.
2. For each x_i, we have two numbers y_i and z_i, where y_i for the positive and z_i for the negative literals.
3. The decimal representation of these numbers is in two parts.
   - The left-hand part comprises a 1 followed by $\ell - i$ 0s.
   - The right-hand part contains one digit for each clause,
     - the digit of y_i in column c_j is 1 iff clause c_j contains literal x_i
     - the digit of z_i in column c_j is 1 iff clause c_j contains literal $\bar{x}_i$
4. The target $t = \underbrace{1 \ldots 1}_{\ell \text{ times}}\underbrace{3 \ldots 3}_{k \text{ times}}$.

Proof (2)

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$					$\ddots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_l						1	0	0	$\cdots$	0
z_l						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$									$\ddots$	$\vdots$
g_k										1
h_k										1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

$$\varphi = \left(x_1 \vee \bar{x}_2 \vee x_3\right) \wedge \left(x_2 \vee x_3 \vee \dots\right) \wedge \dots \wedge \left(\bar{x}_3 \vee \dots \vee \dots\right).$$