Fundamental Computer Science

Denis Trystram
MoSIG1 – University Grenoble-Alpes

February 10, 2020

Summary of previous lecture

- Turing Machines
  - universal computational model
  - all variants of the model are equivalent w.r.t. *decidability*

Complement

- Non-deterministic Turing Machines
  - *decide* the same languages as the deterministic
  - ... but not using the same number of steps

Agenda

- Reduction
- Goal: to classify the problems in complexity classes
  - time complexity: number of steps w.r.t. the size of the input
  - space complexity

Agenda

- Reduction
- Goal: to classify the problems in complexity classes
  - time complexity: number of steps w.r.t. the size of the input
  - space complexity

Focus on *decidable* languages (*solvable* problems)

Time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **time complexity class**

$$\mathrm{TIME}(f(n)) = \{L \mid L \text{ is a language } \textit{decided} \text{ by a Turing Machine in } O(f(n)) \text{ time, where } n \text{ is the size of the input}\}$$

Time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **time complexity class**

$\mathrm{TIME}(f(n)) = \{L \mid L$ is a language *decided* by a Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example: $L = \{0^k 1^k \mid k \geq 0\}$

$M_1 =$ "On input w:

1. Scan the tape and *reject* if a 0 is found on the right of a 1.
2. Repeatedly scan the tape deleting each time a single 0 and a single 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

Time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **time complexity class**

$\mathrm{TIME}(f(n)) = \{L \mid L$ is a language *decided* by a Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example: $L = \{0^k 1^k \mid k \geq 0\} \in \mathrm{TIME}(n^2)$

$M_1 =$ "On input w:

1. Scan the tape and *reject* if a 0 is found on the right of a 1.
2. Repeatedly scan the tape deleting each time a single 0 and a single 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

Time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **time complexity class**

$\mathrm{TIME}(f(n)) = \{L \mid L$ is a language *decided* by a Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example: $L = \{0^k 1^k \mid k \geq 0\}$

$M_2 =$ "On input w:

1. Scan the tape and *reject* if a 0 is found on the right of a 1.
2. Repeat:
   2.1 scan the tape deleting every second 0 and then every second 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

Time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **time complexity class**

$\mathrm{TIME}(f(n)) = \{L \mid L$ is a language *decided* by a Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example: $L = \{0^k 1^k \mid k \geq 0\} \in \mathrm{TIME}(n \log_2 n)$

$M_2 =$ "On input w:

1. Scan the tape and *reject* if a 0 is found on the right of a 1.
2. Repeat:
   2.1 scan the tape deleting every second 0 and then every second 1.
3. If no 0's and no 1's remain in the tape then *accept*, else *reject*."

The class P

A Turing Machine $M = (K, \Sigma, \Gamma, \delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \underline{\sqcup} w) \vdash_M^{p(|w|)} C$.

A language is called **polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

The class P

A Turing Machine $M = (K, \Sigma, \Gamma, \delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \sqcup w) \vdash_M^{p(|w|)} C$.

A language is called **polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

P is the class of *polynomially decidable* languages.

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k)$$

Recall: languages vs problems

- Decision problem: a problem with a yes/no answer
- example
  PATH: Given a graph $G = (V, E)$ and two nodes $s, t \in V$, is there a path from s to t?

Recall: languages vs problems

- Decision problem: a problem with a yes/no answer
- example
  PATH: Given a graph $G = (V, E)$ and two nodes $s, t \in V$, is there a path from s to t?
- Is PATH a language?

Recall: languages vs problems

- Decision problem: a problem with a yes/no answer
- example
  PATH: Given a graph $G = (V, E)$ and two nodes $s, t \in V$, is there a path from s to t?
- Is PATH a language? **No**
- How to define the language corresponding to PATH?

Recall: languages vs problems

- Decision problem: a problem with a yes/no answer
- example
  PATH: Given a graph $G = (V, E)$ and two nodes $s, t \in V$, is there a path from s to t?
- Is PATH a language? **No**
- How to define the language corresponding to PATH?

  $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a graph that has a path from s to $t\}$

  - $\langle G, s, t\rangle$ is the input
  - $|\langle G, s, t\rangle|$ is the size of the input

Recall: languages vs problems

- Decision problem: a problem with a yes/no answer
- example
  PATH: Given a graph $G = (V, E)$ and two nodes $s, t \in V$, is there a path from s to t?
- Is PATH a language? **No**
- How to define the language corresponding to PATH?

  $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a graph that has a path from s to $t\}$

  - $\langle G, s, t\rangle$ is the input
  - $|\langle G, s, t\rangle|$ is the size of the input
- $\mathrm{PATH} \in \mathrm{P}$?

Recall: languages vs problems

- Decision problem: a problem with a yes/no answer
- example
  PATH: Given a graph $G = (V, E)$ and two nodes $s, t \in V$, is there a path from s to t?
- Is PATH a language? **No**
- How to define the language corresponding to PATH?

  $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a graph that has a path from s to $t\}$

  - $\langle G, s, t\rangle$ is the input
  - $|\langle G, s, t\rangle|$ is the size of the input
- $\mathrm{PATH} \in \mathrm{P}$?
  - Yes (i.e., Breadth First Search in $O(|V| + |E|)$)

Enhanced Turing Machine models

- Does the definition of the class P remains the same if we use multiple tapes?

Enhanced Turing Machine models

- Does the definition of the class P remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w|+t)\big)$ steps.

Enhanced Turing Machine models

- Does the definition of the class P remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w| + t)\big)$ steps.

Example: $L = \{0^k 1^k \mid k \geq 0\}$

Enhanced Turing Machine models

- Does the definition of the class P remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w| + t)\big)$ steps.

Example: $L = \{0^k 1^k \mid k \geq 0\}$

$M_3 =$ "On input w:

1. Scan the tape and *reject* if a 0 is found on the right of a 1.
2. Copy the 0's in tape 2.
3. Scan tapes 1 & 2 simultaneously and delete a single 0 from tape 2 and a single 1 from tape 1.
4. If no 0's and no 1's remain then *accept*, else *reject*."

Enhanced Turing Machine models

- Does the definition of the class P remains the same if we use multiple tapes? **YES**
  - Recall: if a multiple tape Turing Machine *halts* on input w after t steps, then the corresponding single tape Turing Machine *halts* after $O\big(t(|w| + t)\big)$ steps.

Example: $L = \{0^k 1^k \mid k \geq 0\}$

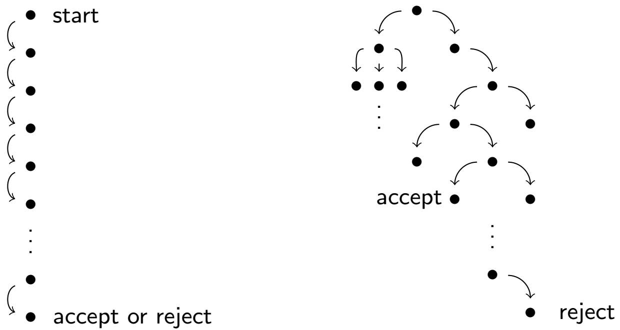
$M_3 =$ "On input w:

1. Scan the tape and *reject* if a 0 is found on the right of a 1.
2. Copy the 0's in tape 2.
3. Scan tapes 1 & 2 simultaneously and delete a single 0 from tape 2 and a single 1 from tape 1.
4. If no 0's and no 1's remain then *accept*, else *reject*."

- complexity: $O(n) \quad \Rightarrow \quad L \in \mathrm{TIME}(n^2) \quad \Rightarrow \quad L \in \mathrm{P}$

Extension to space complexity

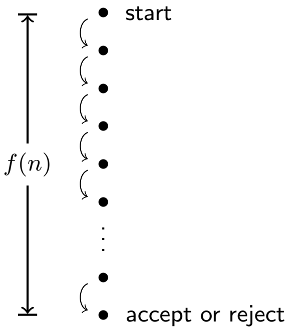
Non-deterministic Turing Machines



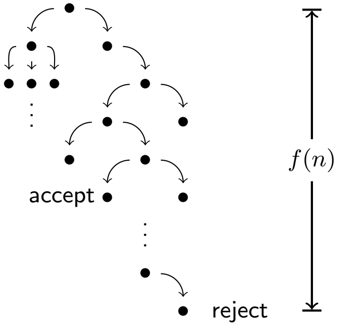
deterministic computation

non-deterministic computation

Non-deterministic Turing Machines



deterministic computation



non-deterministic computation

The **running time** of a non-deterministic Turing Machine which *decides* a language is a function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps on any branch of the computation on any input of length n.

Non-deterministic vs Deterministic Turing Machines

Theorem

Every $f(n)$ time non-deterministic Turing Machine $NDTM$ has an equivalent $2^{O(f(n))}$ time deterministic Turing Machine DTM.

Proof:

- Starting from $NDTM$, construct a 3-tapes DTM

tape 1: input (never changes)
tape 2: simulation
tape 3: address

Non-deterministic vs Deterministic Turing Machines

Theorem

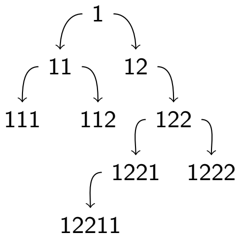
Every $f(n)$ time non-deterministic Turing Machine $NDTM$ has an equivalent $2^{O(f(n))}$ time deterministic Turing Machine DTM.

Proof:

- Starting from $NDTM$, construct a 3-tapes DTM

tape 1: input (never changes)
tape 2: simulation
tape 3: address

- data on tape 3:
  - each node of the computation tree of $NDTM$ has at most c children: $c \leq \Theta(|K|)$
  - address of a node in $\{1, 2, \ldots, c\}^*$



Non-deterministic vs Deterministic Turing Machines

Simulation:

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

Running time

- recall: $c \leq \Theta(|K|)$
- how many nodes in the computation tree?

Non-deterministic vs Deterministic Turing Machines

Simulation:

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

Running time

- recall: $c \leq \Theta(|K|)$
- how many nodes in the computation tree?
  $1 + c + c^2 + \ldots + c^{f(n)} = O(c^{f(n)})$

Non-deterministic vs Deterministic Turing Machines

Simulation:

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

Running time

- recall: $c \leq \Theta(|K|)$
- how many nodes in the computation tree?
  $$1 + c + c^2 + \ldots + c^{f(n)} = O(c^{f(n)})$$
- time to simulate each node: $O\big(f(n)\big)$

Non-deterministic vs Deterministic Turing Machines

Simulation:

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

Running time

- recall: $c \leq \Theta(|K|)$
- how many nodes in the computation tree?
  $$1 + c + c^2 + \ldots + c^{f(n)} = O(c^{f(n)})$$
- time to simulate each node: $O\big(f(n)\big)$
- in total $O(f(n) \cdot c^{f(n)}) = c^{O(f(n))}$

Non-deterministic vs Deterministic Turing Machines

Simulation:

1. Initialize tape 1 with the input w and tapes 2 & 3 to be empty.
2. Copy the contents of tape 1 to tape 2.
3. Simulate NDTM on tape 2 using the sequence of computations described in tape 3. If an accepting configuration is yielded, then *accept*.
4. Update the string in tape 3 with the lexicographic next string and go to 2.

Running time

- recall: $c \leq \Theta(|K|)$
- how many nodes in the computation tree?
  $$1 + c + c^2 + \ldots + c^{f(n)} = O(c^{f(n)})$$
- time to simulate each node: $O\big(f(n)\big)$
- in total $O(f(n) \cdot c^{f(n)}) = c^{O(f(n))}$
- transformation to single tape: $\left(c^{O(f(n))}\right)^2 = c^{O(f(n))}$

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$$\mathrm{NTIME}(f(n)) = \{L \mid L \text{ is a language } \textit{decided} \text{ by a non-deterministic Turing Machine in } O(f(n)) \text{ time, where } n \text{ is the size of the input}\}$$

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$\mathrm{NTIME}(f(n)) = \{L \mid L$ is a language *decided* by a non-deterministic Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example: $\mathrm{COMPOSITES} = \{x \mid x = p \cdot q$, for some integers $p, q > 1\}$

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$$\mathrm{NTIME}(f(n)) = \{L \mid L \text{ is a language } \textit{decided} \text{ by a non-deterministic Turing Machine in } O(f(n)) \text{ time, where } n \text{ is the size of the input}\}$$

Example: $\mathrm{COMPOSITES} = \{x \mid x = p \cdot q$, for some integers $p, q > 1\}$

$M =$ "On input x:

1. Non-deterministically generate two integers $p, q \in [2, \sqrt{x}]$.
2. Compute the product $p \cdot q$
3. If $x = p \cdot q$ then *accept*, else *reject*."

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$\mathrm{NTIME}(f(n)) = \{L \mid L$ is a language *decided* by a non-deterministic Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example: $\mathrm{COMPOSITES} = \{x \mid x = p \cdot q$, for some integers $p, q > 1\}$

$M =$ "On input x:

1. Non-deterministically generate two integers $p, q \in [2, \sqrt{x}]$.
2. Compute the product $p \cdot q$
3. If $x = p \cdot q$ then *accept*, else *reject*."

- M decides $\mathrm{COMPOSITES}$
- $f(n) = O\big(n \cdot \log_2 n \cdot 2^{O(\log_2^* n)}\big)$ (Fürer's algorithm for multiplication)

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$\mathrm{NTIME}(f(n)) = \{L \mid L$ is a language *decided* by a non-deterministic Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example:

$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$\mathrm{NTIME}(f(n)) = \{L \mid L$ is a language *decided* by a non-deterministic Turing Machine in $O(f(n))$ time, where n is the size of the input$\}$

Example:

$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

$M =$ "On input $\langle G, s, t\rangle$:

1. Non-deterministically generate a permutation of the vertex set, $v_1, v_2, \ldots, v_n$.
2. If $v_1 = s$, $v_n = t$ and $(v_i, v_{i+1}) \in E$ for each $i = 1, 2, \ldots n-1$, then *accept*, else *reject*."

Non-deterministic time complexity class

Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We define the **non-deterministic time complexity class**

$$\mathrm{NTIME}(f(n)) = \{L \mid L \text{ is a language } \textit{decided} \text{ by a non-deterministic Turing Machine in } O(f(n)) \text{ time, where } n \text{ is the size of the input}\}$$

Example:

$$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G \text{ is a graph with a Hamiltonian path from } s \text{ to } t\}$$

$M =$ "On input $\langle G, s, t\rangle$:

1. Non-deterministically generate a permutation of the vertex set, $v_1, v_2, \ldots, v_n$.
2. If $v_1 = s$, $v_n = t$ and $(v_i, v_{i+1}) \in E$ for each $i = 1, 2, \ldots n-1$, then *accept*, else *reject*."

- M decides HPATH
- $f(n) = O(n^2) \quad \Rightarrow \quad \mathrm{HPATH} \in \mathrm{NTIME}(n^2)$

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**
- Examples of certificates
  - $\mathrm{COMPOSITES}$: $\langle p, q \rangle$ such $x = p \cdot q$
  - HPATH: $\langle v_1, v_2, \ldots, v_n \rangle$ such that $s = v_1 \to v_2 \to \ldots \to v_n = t$ is a Hamiltonian path from s to t

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**

- Examples of certificates
  - $\mathrm{COMPOSITES}$: $\langle p, q \rangle$ such $x = p \cdot q$
  - HPATH: $\langle v_1, v_2, \ldots, v_n \rangle$ such that $s = v_1 \to v_2 \to \ldots \to v_n = t$ is a Hamiltonian path from s to t

- A **verifier** for a language L is an algorithm $\mathcal{V}$ where

$$L = \{w \mid \mathcal{V} \text{ accepts } \langle w, c \rangle \text{ for each certificate } c\}$$

Certificates and Verifiers

- "*non-deterministically generate*" a string
- check if the generated string has a certain property of the language
- if this input is in the language, then at least one such string exists
- we call this string a **certificate**
- Examples of certificates
  - $\mathrm{COMPOSITES}$: $\langle p, q \rangle$ such $x = p \cdot q$
  - HPATH: $\langle v_1, v_2, \ldots, v_n \rangle$ such that $s = v_1 \to v_2 \to \ldots \to v_n = t$ is a Hamiltonian path from s to t
- A **verifier** for a language L is an algorithm $\mathcal{V}$ where

$$L = \{w \mid \mathcal{V} \text{ accepts } \langle w, c \rangle \text{ for each certificate } c\}$$

- A **polynomial time verifier** runs in polynomial time with respect to the length of the input w

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: $(\Rightarrow)$ Consider a polynomial time verifier $\mathcal{V}$ for L

$NDTM =$

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: ($\Rightarrow$) Consider a polynomial time verifier $\mathcal{V}$ for L

$NDTM =$ "On input w of length n:

1. Non-deterministically generate a string c of length n^k.
2. Run $\mathcal{V}$ on input $\langle w, c \rangle$.
3. If $\mathcal{V}$ accepts, then *accept*, else *reject*."

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: $(\Leftarrow)$ Consider a polynomial time Non-deterministic Turing Machine $NDTM$ that *decides* L

$\mathcal{V} =$

Equivalence of Verifiers and Non-deterministic TM

Theorem

A language L has a polynomial time verifier $\mathcal{V}$ if and only if there is a polynomial time Non-deterministic Turing Machine $NDTM$ which decides it.

Proof: ($\Leftarrow$) Consider a polynomial time Non-deterministic Turing Machine $NDTM$ that *decides* L

$\mathcal{V}$ = "On input $\langle w, c \rangle$:

1. Simulate $NDTM$ on input w using each symbol of c as the non-deterministically choice in order to decide the next step.
2. If this branch of computation accepts, then *accept*, else *reject*."

The class NP

A non-deterministic Turing Machine $M = (K, \Sigma, \Gamma, \Delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \underline{\sqcup} w) \vdash_M^{p(|w|)} C$.

A language is called **non-deterministically polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

The class NP

A non-deterministic Turing Machine $M = (K, \Sigma, \Gamma, \Delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \sqcup w) \vdash_M^{p(|w|)} C$.

A language is called **non-deterministically polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

NP is the class of *non-deterministic polynomially decidable* languages.

$$\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k)$$

The class NP

A non-deterministic Turing Machine $M = (K, \Sigma, \Gamma, \Delta, s, H)$ is called **polynomially bounded** if there is a polynomial p and for any input w there is no configuration C such that $(s, \underline{\sqcup} w) \vdash_M^{p(|w|)} C$.

A language is called **non-deterministically polynomially decidable** if there is a polynomially bounded Turing Machine that *decides* it.

NP is the class of *non-deterministic polynomially decidable* languages.

$$\mathrm{NP} = \bigcup_k \mathrm{NTIME}(n^k)$$

equivalently

NP is the class of languages that have a polynomial time verifier.

P versus NP

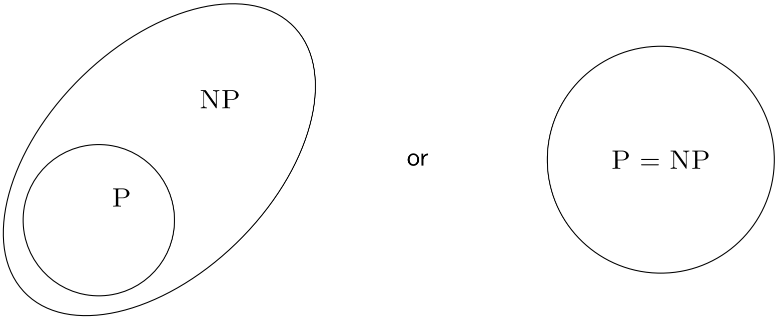
Be careful !!

NP means “non-deterministic polynomial” and not “non-polynomial”

P versus NP

Be careful !!

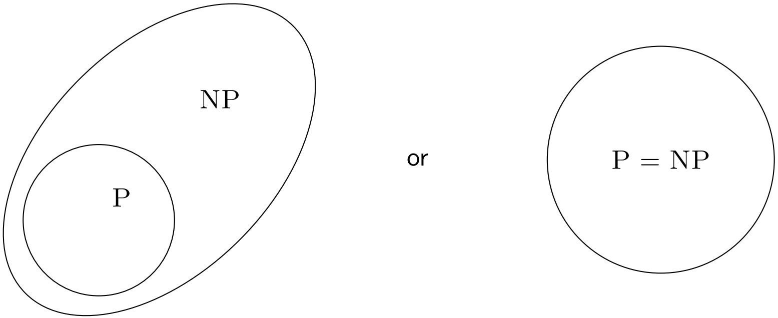
NP means "non-deterministic polynomial" and not "non-polynomial"



P versus NP

Be careful !!

NP means “non-deterministic polynomial” and not “non-polynomial”



What do we know? $\quad \mathrm{NP} \subseteq \mathrm{EXPTIME} = \bigcup_k \mathrm{TIME}(2^{n^k})$

Reductions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is called **polynomial time computable** if there is a polynomially bounded Turing Machine that computes it.

Reductions

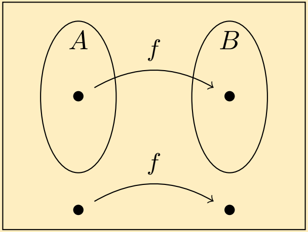
Definition

A function $f : \Sigma^* \to \Sigma^*$ is called **polynomial time computable** if there is a polynomially bounded Turing Machine that computes it.

A language A is **polynomial time reducible** to language B, denoted $A \leq_{\mathrm{P}} B$, if there is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$, where for every input w, it holds that

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

This function f is called a **polynomial time reduction** from A to B.



Reductions

Theorem

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

Proof:

Reductions

Theorem

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

Proof:

- M: a polynomially bounded Turing Machine deciding B
- f: a polynomial time reduction from A to B
- Create a polynomially bounded Turing Machine N deciding A

Reductions

Theorem

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

Proof:

- M: a polynomially bounded Turing Machine deciding B
- f: a polynomial time reduction from A to B
- Create a polynomially bounded Turing Machine N deciding A

  $N =$ "On input w:
  1. Compute $f(w)$.
  2. Run M on $f(w)$ and output whatever M outputs."

Example

$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

$\mathrm{HCYCLE} = \{\langle G\rangle \mid G$ is a graph with a Hamiltonian cycle$\}$

Show that HPATH is polynomial time reducible to HCYCLE.

Solution:

Example

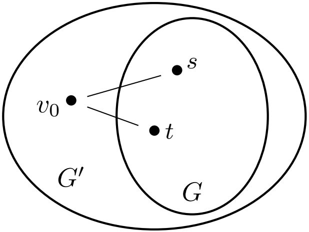
$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

$\mathrm{HCYCLE} = \{\langle G\rangle \mid G$ is a graph with a Hamiltonian cycle$\}$

Show that HPATH is polynomial time reducible to HCYCLE.

Solution:

- input of HPATH: a graph $G = (V, E)$ and two vertices $s, t \in V$
- create an instance of HCYCLE
  - $G' = (V', E')$ where $V' = V \cup \{v_0\}$ and $E' = E \cup \{(v_0, s), (v_0, t)\}$



Example

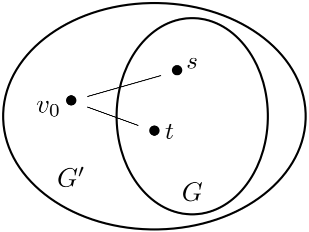
$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

$\mathrm{HCYCLE} = \{\langle G\rangle \mid G$ is a graph with a Hamiltonian cycle$\}$

Show that HPATH is polynomial time reducible to HCYCLE.

Solution:

- input of HPATH: a graph $G = (V, E)$ and two vertices $s, t \in V$
- create an instance of HCYCLE
  - $G' = (V', E')$ where $V' = V \cup \{v_0\}$ and $E' = E \cup \{(v_0, s), (v_0, t)\}$



- The reduction (transformation) is of polynomial time

Example

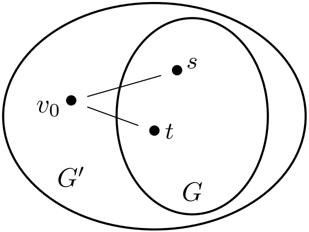
$\mathrm{HPATH} = \{\langle G, s, t\rangle \mid G$ is a graph with a Hamiltonian path from s to $t\}$

$\mathrm{HCYCLE} = \{\langle G\rangle \mid G$ is a graph with a Hamiltonian cycle$\}$

Show that HPATH is polynomial time reducible to HCYCLE.

Solution:

- input of HPATH: a graph $G = (V, E)$ and two vertices $s, t \in V$
- create an instance of HCYCLE
  - $G' = (V', E')$ where $V' = V \cup \{v_0\}$ and $E' = E \cup \{(v_0, s), (v_0, t)\}$



- The reduction (transformation) is of polynomial time

We are not done!!!

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

$(\Rightarrow)$

- consider a Hamiltonian Path from s to t in G:
  $s \to v_2 \to \ldots \to v_{n-1} \to t$

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

($\Rightarrow$)

- consider a Hamiltonian Path from s to t in G:
  $s \to v_2 \to \ldots \to v_{n-1} \to t$
- then $v_0 \to s \to v_2 \to \ldots \to v_{n-1} \to t \to v_0$ is a Hamiltonian Cycle in G'

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

($\Rightarrow$)

- consider a Hamiltonian Path from s to t in G:
  $s \to v_2 \to \ldots \to v_{n-1} \to t$
- then $v_0 \to s \to v_2 \to \ldots \to v_{n-1} \to t \to v_0$ is a Hamiltonian Cycle in G'

($\Leftarrow$)

- consider a Hamiltonian Cycle in G'

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

($\Rightarrow$)

- consider a Hamiltonian Path from s to t in G:
  $s \rightarrow v_2 \rightarrow \ldots \rightarrow v_{n-1} \rightarrow t$
- then $v_0 \rightarrow s \rightarrow v_2 \rightarrow \ldots \rightarrow v_{n-1} \rightarrow t \rightarrow v_0$ is a Hamiltonian Cycle in G'

($\Leftarrow$)

- consider a Hamiltonian Cycle in G'
- this cycle should pass from v_0

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

($\Rightarrow$)

- consider a Hamiltonian Path from s to t in G:
  $s \to v_2 \to \ldots \to v_{n-1} \to t$
- then $v_0 \to s \to v_2 \to \ldots \to v_{n-1} \to t \to v_0$ is a Hamiltonian Cycle in G'

($\Leftarrow$)

- consider a Hamiltonian Cycle in G'
- this cycle should pass from v_0
- there are only two edges incident to v_0: (s, v_0) and (t, v_0)

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

$(\Rightarrow)$

- consider a Hamiltonian Path from s to t in G:
  $s \to v_2 \to \ldots \to v_{n-1} \to t$
- then $v_0 \to s \to v_2 \to \ldots \to v_{n-1} \to t \to v_0$ is a Hamiltonian Cycle in G'

$(\Leftarrow)$

- consider a Hamiltonian Cycle in G'
- this cycle should pass from v_0
- there are only two edges incident to v_0: (s, v_0) and (t, v_0)
- both (s, v_0) and (t, v_0) should be part of the Hamiltonian Cycle

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

($\Rightarrow$)

- consider a Hamiltonian Path from s to t in G:
  $s \to v_2 \to \ldots \to v_{n-1} \to t$
- then $v_0 \to s \to v_2 \to \ldots \to v_{n-1} \to t \to v_0$ is a Hamiltonian Cycle in G'

($\Leftarrow$)

- consider a Hamiltonian Cycle in G'
- this cycle should pass from v_0
- there are only two edges incident to v_0: (s, v_0) and (t, v_0)
- both (s, v_0) and (t, v_0) should be part of the Hamiltonian Cycle
- Hamiltonian Cycle in G': $t \to v_0 \to s \to \ldots \to t$

Example

Solution (cont'd):

There is a Hamiltonian Path from s to t in G if and only if there is a Hamiltonian Cycle in G'

($\Rightarrow$)

- consider a Hamiltonian Path from s to t in G:
  $s \to v_2 \to \ldots \to v_{n-1} \to t$
- then $v_0 \to s \to v_2 \to \ldots \to v_{n-1} \to t \to v_0$ is a Hamiltonian Cycle in G'

($\Leftarrow$)

- consider a Hamiltonian Cycle in G'
- this cycle should pass from v_0
- there are only two edges incident to v_0: (s, v_0) and (t, v_0)
- both (s, v_0) and (t, v_0) should be part of the Hamiltonian Cycle
- Hamiltonian Cycle in G': $t \to v_0 \to s \to \ldots \to t$
- there is a Hamiltonian Path from s to t in G $\square$

Steps of a reduction

Reduction from A to B

1. transform an instance I_{A} of A to an instance I_{B} of B
2. show that the reduction is of polynomial size
3. prove that:
   "there is a solution for the problem A on the instance I_{A}
   if and only if
   there is a solution for the problem B on the instance I_{B}"

Steps of a reduction

Reduction from A to B

1. transform an instance I_{A} of A to an instance I_{B} of B
2. show that the reduction is of polynomial size
3. prove that:
   "there is a solution for the problem A on the instance I_{A}
   if and only if
   there is a solution for the problem B on the instance I_{B}"

Comments

- usually the one direction is trivial (due to the transformation)
- $|I_{\mathrm{B}}|$ is polynomially bounded by $|I_{\mathrm{A}}|$

List of problems

$\mathrm{DIRHCYCLE} = \{\langle G \rangle \mid G$ is a directed graph with a Hamiltonian cycle$\}$

$\mathrm{CLIQUE} = \{\langle G, k \rangle \mid G$ is a graph with a k-clique$\}$

$\mathrm{VERTEX\text{-}COVER} = \{\langle G, k \rangle \mid G$ is a graph with a set $A \subseteq V$ such that $|A| = k$ and every $e \in E$ is incident to a vertex in $A\}$

$\mathrm{INDEPENDENT\text{-}SET} = \{\langle G, k \rangle \mid G$ is a graph with a set $A \subseteq V$ such that $|A| = k$ and there is no edge between any pair of vertices in $A\}$

$\mathrm{LONGEST\text{-}PATH} = \{\langle G, s, t, k \rangle \mid G$ is a graph with a path from s to t of length at least $k\}$

Exercises

- Show that HCYCLE is polynomial time reducible to HPATH.