Complexity classes

Peter Hajnal

Bolyai Institute, SZTE, Szeged

2023 fall

Cpmlexity of a Turing machine/algorithm on an input

Cpmlexity of a Turing machine/algorithm on an input

Definition

The *time complexity* of a Turing machin T on an ω input is $\ell := TIME(\omega, T)$ if its truncated run is $(\kappa_i)_{i=0}^{\ell}$, and ∞ if the run is an infinite loop (it does not reach the *STOP* state).

Cpmlexity of a Turing machine/algorithm on an input

Definition

The *time complexity* of a Turing machin T on an ω input is $\ell := TIME(\omega, T)$ if its truncated run is $(\kappa_i)_{i=0}^{\ell}$, and ∞ if the run is an infinite loop (it does not reach the *STOP* state).

We will also use this notion for Turing machines, designed for decision making. In this case the role of *STOP* will be substituted by *ACCEPT*/*REJECT*.

Cpmlexity of a Turing machine/algorithm on an input

Definition

The *time complexity* of a Turing machin T on an ω input is $\ell := TIME(\omega, T)$ if its truncated run is $(\kappa_i)_{i=0}^{\ell}$, and ∞ if the run is an infinite loop (it does not reach the *STOP* state).

We will also use this notion for Turing machines, designed for decision making. In this case the role of *STOP* will be substituted by *ACCEPT*/*REJECT*.

Definition

The *space complexity* of a Turing-machine T on an input ω is $s := SPACE(\omega, T)$, if during its execution on ω the largest index of the cells under the work hand is s, or ∞, if the working eye/hand moves arbitrarily far from the left border of the work tape.

An Observation

An Observation

The Turing machine can visit at most as many cells on the working tape as many times it moves.

An Observation

The Turing machine can visit at most as many cells on the working tape as many times it moves.

Observation

$$SPACE(\omega, T) \leq TIME(\omega, T)$$

Complexity of a machine/algorithm

Complexity of a machine/algorithm

Definition

Let $t : \mathbb{N} \to \mathbb{R}$ be an arbitrary function. We say that a Turing machine T is an element of the set $\texttt{TIME}(t(n))$ if for every $\omega \in \Sigma^*$

$$TIME(\omega, T) \leq t(|\omega|).$$

Complexity of a machine/algorithm

Definition

Let $t : \mathbb{N} \to \mathbb{R}$ be an arbitrary function. We say that a Turing machine T is an element of the set $\texttt{TIME}(t(n))$ if for every $\omega \in \Sigma^*$

$$TIME(\omega, T) \leq t(|\omega|).$$

Definition

Let $s : \mathbb{N} \to \mathbb{R}$ be an arbitrary function. We say that a Turing machine T is an element of the set $\texttt{SPACE}(s(n))$ if for every $\omega \in \Sigma^*$

$$SPACE(\omega, T) \leq s(|\omega|).$$

Classes of languages

Classes of languages

Definition

Let $t : \mathbb{N} \to \mathbb{R}$ be an arbitrary function. We say that a language L is an element of the set $\mathcal{TIME}(t(n))$, if there exists a Turing machine T such that

(i) decides L, and

(ii)) $T \in \mathtt{TIME}(t(n))$, i.e. $TIME(\omega, T) \leq t(|\omega|)$.

Classes of languages

Definition

Let $t : \mathbb{N} \to \mathbb{R}$ be an arbitrary function. We say that a language L is an element of the set $\mathcal{TIME}(t(n))$, if there exists a Turing machine T such that

(i) decides L, and

(ii)) $T \in \mathtt{TIME}(t(n))$, i.e. $TIME(\omega, T) \leq t(|\omega|)$.

Definition

Let $s : \mathbb{N} \to \mathbb{R}$ be an arbitrary function. We say that a language L is an element of the set $\mathcal{SPACE}(s(n))$, if there exists a Turing machine T such that

(i) decides L, and

(ii) $T \in \mathtt{SPACE}(s(n))$, i.e. $SPACE(\omega, T) \leq s(|\omega|)$.

$\mathcal{P}$

$\mathcal{P}$

The classes $\mathcal{TIME}(t(n))/\mathcal{SPACE}(s(n))$ are very much depend on the model we use. We obtain much more useful classes when time or space constraint is not specified by a function, but by an "order of magnitude".

$\mathcal{P}$

The classes $\mathcal{TIME}(t(n))/\mathcal{SPACE}(s(n))$ are very much depend on the model we use. We obtain much more useful classes when time or space constraint is not specified by a function, but by an "order of magnitude".

Definition: Decidable languages in polynomial time

$$\mathcal{P} := \bigcup_{p \in \mathbb{R}[x]} \mathcal{TIME}(p(n)) = \bigcup_{a \in \mathbb{N}} \mathcal{TIME}(an^a + a).$$

$\mathcal{EXP}, \mathcal{PSPACE}$

$\mathcal{EXP}, \mathcal{PSPACE}$

Definition: Decidable languages in exponential time

$$\mathcal{EXP} := \bigcup_{a\in\mathbb{N}} \mathcal{TIME}(2^{an^a+a}).$$

$\mathcal{EXP}, \mathcal{PSPACE}$

Definition: Decidable languages in exponential time

$$\mathcal{EXP} := \bigcup_{a\in\mathbb{N}} \mathcal{TIME}(2^{an^a+a}).$$

Definition: Decidable languages in polynomial space

$$\mathcal{PSPACE} := \bigcup_{a\in\mathbb{N}} \mathcal{SPACE}(an^a + a).$$

$\mathcal{EXPSPACE}, \mathcal{L}$

$\mathcal{EXPSPACE}, \mathcal{L}$

Definition: Decidable languages in exponential space

$$\mathcal{EXPSPACE} := \bigcup_{a\in\mathbb{N}} \mathcal{SPACE}(2^{an^a+a}).$$

$\mathcal{EXPSPACE}, \mathcal{L}$

Definition: Decidable languages in exponential space

$$\mathcal{EXPSPACE} := \bigcup_{a\in\mathbb{N}} \mathcal{SPACE}(2^{an^a+a}).$$

Definition: Decidable languages in logarithmic space

$$\mathcal{L} := \bigcup_{a\in\mathbb{N}} \mathcal{SPACE}(a\log(n+2)).$$

$\mathcal{EXPSPACE}, \mathcal{L}$

Definition: Decidable languages in exponential space

$$\mathcal{EXPSPACE} := \bigcup_{a\in\mathbb{N}} \mathcal{SPACE}(2^{an^a+a}).$$

Definition: Decidable languages in logarithmic space

$$\mathcal{L} := \bigcup_{a\in\mathbb{N}} \mathcal{SPACE}(a\log(n+2)).$$

The complete list of complexity classes is much longer
http://qwiki.stanford.edu/index.php/Complexity_Zoo.

Break

Inclusions

Inclusions

The following inclusions are straight forward:

Inclusions

The following inclusions are straight forward:

$$\begin{array}{ccccccccccc} \mathcal{L} & \subseteq & \mathcal{PSPACE} & \subseteq & \mathcal{EXPSPACE} & \subseteq & \mathcal{D} & \subseteq & \mathcal{S} & \subseteq & \mathcal{P}(\Sigma^*) \\ & & \cup\!\mid & & \cup\!\mid & & & & & & \\ & & \mathcal{P} & \subseteq & \mathcal{EXP} & & & & & & \end{array}$$

Inclusions

The following inclusions are straight forward:

$$\begin{array}{ccccccccccc} \mathcal{L} & \subseteq & \mathcal{PSPACE} & \subseteq & \mathcal{EXPSPACE} & \subseteq & \mathcal{D} & \subseteq & \mathcal{S} & \subseteq & \mathcal{P}(\Sigma^*) \\ & & \cup\!\mid & & \cup\!\mid & & & & & & \\ & & \mathcal{P} & \subseteq & \mathcal{EXP} & & & & & & \end{array}$$

Our goal is to prove that:

$$\mathcal{L} \subseteq \mathcal{P} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP} \subseteq \mathcal{EXPSPACE} \subseteq \mathcal{D} \subseteq \mathcal{S} \subseteq \mathcal{P}(\Sigma^*).$$

Inclusions

The following inclusions are straight forward:

$$\begin{array}{ccccccccccc} \mathcal{L} & \subseteq & \mathcal{PSPACE} & \subseteq & \mathcal{EXPSPACE} & \subseteq & \mathcal{D} & \subseteq & \mathcal{S} & \subseteq & \mathcal{P}(\Sigma^*) \\ & & \cup\!\mid & & \cup\!\mid & & & & & & \\ & & \mathcal{P} & \subseteq & \mathcal{EXP} & & & & & & \end{array}$$

Our goal is to prove that:

$$\mathcal{L} \subseteq \mathcal{P} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP} \subseteq \mathcal{EXPSPACE} \subseteq \mathcal{D} \subseteq \mathcal{S} \subseteq \mathcal{P}(\Sigma^*).$$

To prove these inclusions the following Lemma will be useful:

Inclusions

The following inclusions are straight forward:

$$\begin{array}{ccccccccccc} \mathcal{L} & \subseteq & \mathcal{PSPACE} & \subseteq & \mathcal{EXPSPACE} & \subseteq & \mathcal{D} & \subseteq & \mathcal{S} & \subseteq & \mathcal{P}(\Sigma^*) \\ & & \cup\!| & & \cup\!| & & & & & & \\ & & \mathcal{P} & \subseteq & \mathcal{EXP} & & & & & & \end{array}$$

Our goal is to prove that:

$$\mathcal{L} \subseteq \mathcal{P} \subseteq \mathcal{PSPACE} \subseteq \mathcal{EXP} \subseteq \mathcal{EXPSPACE} \subseteq \mathcal{D} \subseteq \mathcal{S} \subseteq \mathcal{P}(\Sigma^*).$$

To prove these inclusions the following Lemma will be useful:

Lemma

Lemma

$$\mathcal{SPACE}(s(n)) \subseteq \cup_{c\in\mathbb{N}} \mathcal{TIME}(c^{s(n)+\log(n+1)}).$$

The proof of the Lemma I

The proof of the Lemma I

Let L be a language in $\mathcal{SPACE}(s(n))$. Then there is a Turing machine T, that decides L (specially it stops on all $\omega \in L$), and its space complexity is at most $s(n)$.

The proof of the Lemma I

Let L be a language in $\mathcal{SPACE}(s(n))$. Then there is a Turing machine T, that decides L (specially it stops on all $\omega \in L$), and its space complexity is at most $s(n)$.

Notation

The previous sentence is a characteristic first line in proofs on complexity. We introduce a spacial notation for that: $L \in_T \mathcal{SPACE}(s(n))$.

The proof of the Lemma I

Let L be a language in $\mathcal{SPACE}(s(n))$. Then there is a Turing machine T, that decides L (specially it stops on all $\omega \in L$), and its space complexity is at most $s(n)$.

Notation

The previous sentence is a characteristic first line in proofs on complexity. We introduce a spacial notation for that:
$L \in_T \mathcal{SPACE}(s(n))$.

Let $\kappa_0(\omega) \to \kappa_1(\omega) \to \kappa_2(\omega) \to \ldots \to \kappa_\ell(\omega)$ be the run on input ω. This is a sequence of configurations of length $\ell \geq 1$, where the first one ($\kappa_0(\omega)$) is the initial configuration (specially the state is START) and the last configuration is ($\kappa_\ell(\omega)$) the first one, where the state is ACCEPT or REJECT.

The proof of the Lemma II

The proof of the Lemma II

It is easy to see that the configurations in the sequence must be different, i.e. if $i \neq j$ then $\kappa_i \neq \kappa_j$.

The proof of the Lemma II

It is easy to see that the configurations in the sequence must be different, i.e. if $i \neq j$ then $\kappa_i \neq \kappa_j$.

Let $n := |\omega|$. **Question:** How many different configurations can we imagine?

The proof of the Lemma II

It is easy to see that the configurations in the sequence must be different, i.e. if $i \neq j$ then $\kappa_i \neq \kappa_j$.

Let $n := |\omega|$. **Question:** How many different configurations can we imagine?

An upper bound on the answer is:

$$(n+2) \cdot (s(n)+1) \cdot |\Gamma|^{s(n)} \cdot |S| \leq \alpha_T (n+1) \alpha_T^{s(n)} \leq \beta_T^{s(n)+\log(n+1)},$$

where α_T, β_T constants depending on T.

The proof of the Lemma II

It is easy to see that the configurations in the sequence must be different, i.e. if $i \neq j$ then $\kappa_i \neq \kappa_j$.

Let $n := |\omega|$. **Question:** How many different configurations can we imagine?

An upper bound on the answer is:

$$(n+2) \cdot (s(n)+1) \cdot |\Gamma|^{s(n)} \cdot |S| \leq \alpha_T(n+1)\alpha_T^{s(n)} \leq \beta_T^{s(n)+\log(n+1)},$$

where α_T, β_T constants depending on T.

Indeed: The number of the possible position of the input eye is $n+2$.

The proof of the Lemma II

It is easy to see that the configurations in the sequence must be different, i.e. if $i \neq j$ then $\kappa_i \neq \kappa_j$.

Let $n := |\omega|$. **Question:** How many different configurations can we imagine?

An upper bound on the answer is:

$$(n+2) \cdot (s(n)+1) \cdot |\Gamma|^{s(n)} \cdot |S| \leq \alpha_T (n+1) \alpha_T^{s(n)} \leq \beta_T^{s(n)+\log(n+1)},$$

where α_T, β_T constants depending on T.

Indeed: The number of the possible position of the input eye is $n+2$. The number of the possible position of the work eye/pen is $s(n)+1$.

The proof of the Lemma II

It is easy to see that the configurations in the sequence must be different, i.e. if $i \neq j$ then $\kappa_i \neq \kappa_j$.

Let $n := |\omega|$. **Question:** How many different configurations can we imagine?

An upper bound on the answer is:

$$(n+2) \cdot (s(n)+1) \cdot |\Gamma|^{s(n)} \cdot |S| \leq \alpha_T (n+1) \alpha_T^{s(n)} \leq \beta_T^{s(n)+\log(n+1)},$$

where α_T, β_T constants depending on T.

Indeed: The number of the possible position of the input eye is $n+2$. The number of the possible position of the work eye/pen is $s(n)+1$. The number of the possible content of the work tape is $(|\Gamma|+1)^{s(n)}$.

The proof of the Lemma II

It is easy to see that the configurations in the sequence must be different, i.e. if $i \neq j$ then $\kappa_i \neq \kappa_j$.

Let $n := |\omega|$. **Question:** How many different configurations can we imagine?

An upper bound on the answer is:

$$(n+2)\cdot(s(n)+1)\cdot|\Gamma|^{s(n)}\cdot|S| \leq \alpha_T(n+1)\alpha_T^{s(n)} \leq \beta_T^{s(n)+\log(n+1)},$$

where α_T, β_T constants depending on T.

Indeed: The number of the possible position of the input eye is $n+2$. The number of the possible position of the work eye/pen is $s(n)+1$. The number of the possible content of the work tape is $(|\Gamma|+1)^{s(n)}$. Finally the state of the machine comes from $|S|$ possibilities.

The proof of the Lemma III

The proof of the Lemma III

If the runtime were longer than ${\beta_T}^{s(n)+\log n}$, then some configurations would be repeated during the run, so the run would be infinite.

The proof of the Lemma III

If the runtime were longer than ${\beta_T}^{s(n)+\log n}$, then some configurations would be repeated during the run, so the run would be infinite.

We know that this is not the case. So we get the bound claimed.

Robustness

Robustness

The mathematical definition of the Turing machine has a lot of subtleties. Researchers/lecturers/textbook writers have many different definitions, since everyone chooses the most appealing version of the concept of a Turing machine.

Robustness

The mathematical definition of the Turing machine has a lot of subtleties. Researchers/lecturers/textbook writers have many different definitions, since everyone chooses the most appealing version of the concept of a Turing machine.

Nevertheless the possible choices do not affect the concept of computability/decidability.

Robustness

The mathematical definition of the Turing machine has a lot of subtleties. Researchers/lecturers/textbook writers have many different definitions, since everyone chooses the most appealing version of the concept of a Turing machine.

Nevertheless the possible choices do not affect the concept of computability/decidability.

A similar comment applies to the complexity classes defined with constraint on the time/space described by order of magnitude. For example $\mathcal{P}$ (the class of languages that can be decided in polynomial time) remains the same when working with a different model.

Robustness

The mathematical definition of the Turing machine has a lot of subtleties. Researchers/lecturers/textbook writers have many different definitions, since everyone chooses the most appealing version of the concept of a Turing machine.

Nevertheless the possible choices do not affect the concept of computability/decidability.

A similar comment applies to the complexity classes defined with constraint on the time/space described by order of magnitude. For example $\mathcal{P}$ (the class of languages that can be decided in polynomial time) remains the same when working with a different model.

$\mathcal{P}$ is a robust class of languages.

Dangers

Dangers

It would cause a much bigger problem, if we were to define the $\mathcal{LINEAR} := \{L(T) : T \in \cup_{a\in\mathbb{N}}\texttt{Time}(an+a)\}$ language class.

Dangers

It would cause a much bigger problem, if we were to define the $\mathcal{LINEAR} := \{L(T) : T \in \cup_{a\in\mathbb{N}}\texttt{Time}(an+a)\}$ language class.

This language class is not so robust anymore.

Dangers

It would cause a much bigger problem, if we were to define the $\mathcal{LINEAR} := \{L(T) : T \in \cup_{a\in\mathbb{N}}\texttt{Time}(an+a)\}$ language class.

This language class is not so robust anymore.

If we deal with such a class then clarifying/understanding the exact model of computation is crucial.

Dangers

It would cause a much bigger problem, if we were to define the $\mathcal{LINEAR} := \{L(T) : T \in \cup_{a\in\mathbb{N}}\texttt{Time}(an + a)\}$ language class.

This language class is not so robust anymore.

If we deal with such a class then clarifying/understanding the exact model of computation is crucial.

Similarly serious technical problems would be raised by the language class $\mathcal{E} := \{L(T) : T \in \cup_{a\in\mathbb{N}}\texttt{Time}(a^{n+a})\}$.

Break

The language PALINDROM

The language PALINDROM

Definition: PALINDROM language

$$PALINDROM = \{\omega = \omega_1, \ldots \omega_n : \text{where } \omega_i = \omega_{n+1-i} \text{ for all } i \in \{1, 2, \ldots, n\}\}.$$

The language PALINDROM

Definition: PALINDROM language

$$PALINDROM = \{\omega = \omega_1, \ldots \omega_n : \text{where } \omega_i = \omega_{n+1-i} \text{ for all } i \in \{1, 2, \ldots, n\}\}.$$

So we have a decision problem. Given a word ω, we have to decide ω is a palindrom or not. So we don't need an output tape, but the set of states S contains the states REJECT and ACCEPT.

The language PALINDROM

Definition: PALINDROM language

$$PALINDROM = \{\omega = \omega_1, \ldots \omega_n : \text{where } \omega_i = \omega_{n+1-i} \text{ for all } i \in \{1, 2, \ldots, n\}\}.$$

So we have a decision problem. Given a word ω, we have to decide ω is a palindrom or not. So we don't need an output tape, but the set of states S contains the states REJECT and ACCEPT.

Two Turing machines/algorithms are sketched. For simplicity we assume that $\Sigma = \{0, 1\}$.

First model: Single tape model

First model: Single tape model

- The first decision is to choose an alphabet for the work tape: $\Gamma = \{0, 1, 0^{\checkmark}, 1^{\checkmark}\}$.

First algorithm/TM: Hogh level description

First algorithm/TM: Hogh level description

- On the left, find the first unchecked bit.

First algorithm/TM: Hogh level description

- On the left, find the first unchecked bit.

Mark it and remember its value.

First algorithm/TM: Hogh level description

- On the left, find the first unchecked bit.

Mark it and remember its value.

On the right, find the last unchecked bit and mark if it matches.

First algorithm/TM: Hogh level description

- On the left, find the first unchecked bit.

Mark it and remember its value.

On the right, find the last unchecked bit and mark if it matches.

If so, we continue the procedure, until we run out of unchecked bits.

First algorithm/TM: S, set of states

For the Turing machine's work so far, we used the following state set

$$S = \{\text{START, LEFT-MARK, RIGHT-FIND-0, RIGHT-FIND-1, RIGHT-TEST-0, RIGHT-TEST-1, LEFT-FIND, ACCEPT, REJECT}\}.$$

First algorithm/TM: Transition function (fragments)

First algorithm/TM: Transition function (fragments)

$$(START, \triangleright) \mapsto (\text{LEFT-MARK}, *, R)$$

First algorithm/TM: Transition function (fragments)

$$(START, \triangleright) \mapsto (\text{LEFT-MARK}, *, R)$$

$$(\text{LEFT-MARK}, 0) \mapsto (\text{RIGHT-FIND-0}, 0^{\checkmark}, R)$$

First algorithm/TM: Transition function (fragments)

$$(START, \triangleright) \mapsto (\text{LEFT-MARK}, *, R)$$

$$(\text{LEFT-MARK}, 0) \mapsto (\text{RIGHT-FIND-0}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 1) \mapsto (\text{RIGHT-FIND-1}, 1^{\checkmark}, R)$$

First algorithm/TM: Transition function (fragments)

$$(START, \triangleright) \mapsto (\text{LEFT-MARK}, *, R)$$

$$(\text{LEFT-MARK}, 0) \mapsto (\text{RIGHT-FIND-0}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 1) \mapsto (\text{RIGHT-FIND-1}, 1^{\checkmark}, R)$$

$$(\text{RIGHT-FIND-1}, 0) \mapsto (\text{RIGHT-FIND-1}, 0, R)$$

First algorithm/TM: Transition function (fragments)

$$(START, \triangleright) \mapsto (\text{LEFT-MARK}, *, R)$$

$$(\text{LEFT-MARK}, 0) \mapsto (\text{RIGHT-FIND-0}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 1) \mapsto (\text{RIGHT-FIND-1}, 1^{\checkmark}, R)$$

$$(\text{RIGHT-FIND-1}, 0) \mapsto (\text{RIGHT-FIND-1}, 0, R)$$

$$(\text{RIGHT-FIND-1}, \triangleleft) \mapsto (\text{RIGHT-TEST-1}, *, L)$$

First algorithm/TM: Transition function (fragments)

$$(START, \triangleright) \mapsto (\text{LEFT-MARK}, *, R)$$

$$(\text{LEFT-MARK}, 0) \mapsto (\text{RIGHT-FIND-0}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 1) \mapsto (\text{RIGHT-FIND-1}, 1^{\checkmark}, R)$$

$$(\text{RIGHT-FIND-1}, 0) \mapsto (\text{RIGHT-FIND-1}, 0, R)$$

$$(\text{RIGHT-FIND-1}, \triangleleft) \mapsto (\text{RIGHT-TEST-1}, *, L)$$

$$(\text{RIGHT-TEST-1}, 0) \mapsto (REJECT, *, *)$$

First algorithm/TM: Transition function (fragments)

$$(START, \triangleright) \mapsto (\text{LEFT-MARK}, *, R)$$

$$(\text{LEFT-MARK}, 0) \mapsto (\text{RIGHT-FIND-0}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 1) \mapsto (\text{RIGHT-FIND-1}, 1^{\checkmark}, R)$$

$$(\text{RIGHT-FIND-1}, 0) \mapsto (\text{RIGHT-FIND-1}, 0, R)$$

$$(\text{RIGHT-FIND-1}, \triangleleft) \mapsto (\text{RIGHT-TEST-1}, *, L)$$

$$(\text{RIGHT-TEST-1}, 0) \mapsto (REJECT, *, *)$$

$$(\text{RIGHT-TEST-1}, 1) \mapsto (\text{LEFT-FIND}, 1^{\checkmark}, L)$$

FIRST algorithm/TM: Transition function

FIRST algorithm/TM: Transition function

$$(\text{RIGHT-TEST-1}, 1^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

FIRST algorithm/TM: Transition function

$$(\text{RIGHT-TEST-1}, 1^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{RIGHT-TEST-1}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

FIRST algorithm/TM: Transition function

$$(\text{RIGHT-TEST-1}, 1^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{RIGHT-TEST-1}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{LEFT-FIND}, 0) \mapsto (\text{LEFT-FIND}, 0, L)$$

FIRST algorithm/TM: Transition function

$$(\text{RIGHT-TEST-1}, 1^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{RIGHT-TEST-1}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{LEFT-FIND}, 0) \mapsto (\text{LEFT-FIND}, 0, L)$$

$$(\text{LEFT-FIND}, 0^{\checkmark}) \mapsto (\text{LEFT-MARK}, 0^{\checkmark}, R)$$

FIRST algorithm/TM: Transition function

$$(\text{RIGHT-TEST-1}, 1^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{RIGHT-TEST-1}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{LEFT-FIND}, 0) \mapsto (\text{LEFT-FIND}, 0, L)$$

$$(\text{LEFT-FIND}, 0^{\checkmark}) \mapsto (\text{LEFT-MARK}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

FIRST algorithm/TM: Transition function

$$(\text{RIGHT-TEST-1}, 1^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{RIGHT-TEST-1}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{LEFT-FIND}, 0) \mapsto (\text{LEFT-FIND}, 0, L)$$

$$(\text{LEFT-FIND}, 0^{\checkmark}) \mapsto (\text{LEFT-MARK}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{LEFT-FIND}, 0) \mapsto (\text{LEFT-FIND}, 0, L)$$

FIRST algorithm/TM: Transition function

$$(\text{RIGHT-TEST-1}, 1^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{RIGHT-TEST-1}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{LEFT-FIND}, 0) \mapsto (\text{LEFT-FIND}, 0, L)$$

$$(\text{LEFT-FIND}, 0^{\checkmark}) \mapsto (\text{LEFT-MARK}, 0^{\checkmark}, R)$$

$$(\text{LEFT-MARK}, 0^{\checkmark}) \mapsto (\text{ACCEPT}, *, *)$$

$$(\text{LEFT-FIND}, 0) \mapsto (\text{LEFT-FIND}, 0, L)$$

$$(\text{START}, 0) \mapsto \text{„Who cares?''}$$

First algorithm/TG: The movement of the head, time

First algorithm/TG: The movement of the head, time

If ω is a palindrome word, then the above $\mathcal{A}_1$ algorithm/machine run can be easily calculated.

First algorithm/TG: The movement of the head, time

If ω is a palindrome word, then the above $\mathcal{A}_1$ algorithm/machine run can be easily calculated.

The movement of the head is easy to describe.

First algorithm/TG: The movement of the head, time

If ω is a palindrome word, then the above $\mathcal{A}_1$ algorithm/machine run can be easily calculated.

The movement of the head is easy to describe. It will be a *damped pendulum motion*.

First algorithm/TG: The movement of the head, time

If ω is a palindrome word, then the above $\mathcal{A}_1$ algorithm/machine run can be easily calculated.

The movement of the head is easy to describe. It will be a *damped pendulum motion*.

theorem

$$TIME(\mathcal{A}_1, \omega) = \mathcal{O}(|\omega|^2)$$

First algorithm/TG: The movement of the head, time

If ω is a palindrome word, then the above $\mathcal{A}_1$ algorithm/machine run can be easily calculated.

The movement of the head is easy to describe. It will be a *damped pendulum motion*.

theorem

$$\mathit{TIME}(\mathcal{A}_1, \omega) = \mathcal{O}(|\omega|^2)$$

Constants are not calculated. Also irrelevant, by increasing the number of states the running time can be reduced.

First algorithm/TG: The movement of the head, time

If ω is a palindrome word, then the above $\mathcal{A}_1$ algorithm/machine run can be easily calculated.

The movement of the head is easy to describe. It will be a *damped pendulum motion*.

theorem

$$\mathit{TIME}(\mathcal{A}_1, \omega) = \mathcal{O}(|\omega|^2)$$

Constants are not calculated. Also irrelevant, by increasing the number of states the running time can be reduced. For example, we could use states for examples RIGHT-TEST-000, RIGHT-TEST-001, RIGHT-TEST-010, RIGHT-TEST-011, . . .

Second model: Standard model with 1 work tape

Second model: Standard model with 1 work tape

$\Gamma = \Sigma = \{0, 1\}$.

Second algorithm/TM: High level description

Second algorithm/TM: High level description

The input is copied to the working tape.

Second algorithm/TM: High level description

The input is copied to the working tape.

The input head goes to the beginning of the input.

Second algorithm/TM: High level description

The input is copied to the working tape.

The input head goes to the beginning of the input. // The work head is at the end of the copied input.

Second algorithm/TM: High level description

The input is copied to the working tape.

The input head goes to the beginning of the input. // The work head is at the end of the copied input.

The the eyes move in different directions and test the palindrom property.

Second algorithm/TM: S, the set of states

Second algorithm/TM: S, the set of states

$$S = \{\text{START, COPY, TO-THE-LEFT, TEST, ACCEPT, REJECT}\}.$$

Second algorithm/TM: Transition function

Second algorithm/TM: Transition function

$$(START, \triangleright, \triangleright) \mapsto (\mathrm{COPY}, R, *, R)$$

Second algorithm/TM: Transition function

$$(START, \triangleright, \triangleright) \mapsto (\mathrm{COPY}, R, *, R)$$

$$(\mathrm{COPY}, 0, \smile) \mapsto (\mathrm{COPY}, R, 0, R)$$

Second algorithm/TM: Transition function

$$(START, \triangleright, \triangleright) \mapsto (\text{COPY}, R, *, R)$$

$$(\text{COPY}, 0, \smile) \mapsto (\text{COPY}, R, 0, R)$$

$$(\text{COPY}, 1, \smile) \mapsto (\text{COPY}, R, 1, R)$$

Second algorithm/TM: Transition function

$$(START, \triangleright, \triangleright) \mapsto (\text{COPY}, R, *, R)$$

$$(\text{COPY}, 0, \smile) \mapsto (\text{COPY}, R, 0, R)$$

$$(\text{COPY}, 1, \smile) \mapsto (\text{COPY}, R, 1, R)$$

$$(\text{COPY}, \triangleleft, \smile) \mapsto (\text{TO-THE-LEFT}, L, *, L)$$

Second algorithm/TM: Transition function

$$(START, \triangleright, \triangleright) \mapsto (\text{COPY}, R, *, R)$$

$$(\text{COPY}, 0, \smile) \mapsto (\text{COPY}, R, 0, R)$$

$$(\text{COPY}, 1, \smile) \mapsto (\text{COPY}, R, 1, R)$$

$$(\text{COPY}, \triangleleft, \smile) \mapsto (\text{TO-THE-LEFT}, L, *, L)$$

$$(\text{TO-THE-LEFT}, 0, 0) \mapsto (\text{TO-THE-LEFT}, L, 0, .)$$

Second algorithm/TM: Transition function

$$(START, \triangleright, \triangleright) \mapsto (\text{COPY}, R, *, R)$$

$$(\text{COPY}, 0, \smile) \mapsto (\text{COPY}, R, 0, R)$$

$$(\text{COPY}, 1, \smile) \mapsto (\text{COPY}, R, 1, R)$$

$$(\text{COPY}, \triangleleft, \smile) \mapsto (\text{TO-THE-LEFT}, L, *, L)$$

$$(\text{TO-THE-LEFT}, 0, 0) \mapsto (\text{TO-THE-LEFT}, L, 0, .)$$

$$(\text{TO-THE-LEFT}, \triangleright, 0) \mapsto (\text{TEST}, R, 0, .),$$

Second algorithm/TM: Transition function

Second algorithm/TM: Transition function

$$(\text{TEST}, 1, 1) \mapsto (\text{TEST}, R, *, L),$$

Second algorithm/TM: Transition function

$$(\text{TEST}, 1, 1) \mapsto (\text{TEST}, R, *, L),$$

$$(\text{TEST}, 1, 0) \mapsto (\text{REJECT}, *, *, *),$$

Second algorithm/TM: Transition function

$$(\text{TEST}, 1, 1) \mapsto (\text{TEST}, R, *, L),$$

$$(\text{TEST}, 1, 0) \mapsto (\text{REJECT}, *, *, *),$$

$$(\text{TEST}, \triangleleft, \triangleright) \mapsto (\text{ACCEPT}, ., *, .).$$

Second algorithm/TM: The time

Second algorithm/TM: The time

Observation

All ω input, the run length is at most $3|\omega| + 3 = \mathcal{O}(|\omega|)$.

Second algorithm/TM: The time

Observation

All ω input, the run length is at most $3|\omega| + 3 = \mathcal{O}(|\omega|)$.

Specifically

$$TIME(\omega; T) = \Theta(|\omega|)$$

Second algorithm/TM: The time

Observation

All ω input, the run length is at most $3|\omega| + 3 = \mathcal{O}(|\omega|)$.

Specifically

$$TIME(\omega; T) = \Theta(|\omega|)$$

This result is sharp in terms of magnitude.

Break

The one-tape model is bad

Theorem

If T is a Turing machine that decides in the single-tape model the *PALINDROM* language, then $\forall\, n$, $\exists\, \omega \in \Sigma^n$:

$$TIME(\omega, T) \geq \alpha_T |\omega|^2,$$

for some positive constant α_T.

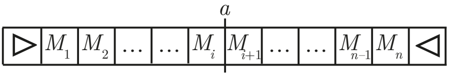
Proof: Notions

Proof: Notions

The common boundary of two adjacent cells on the input tape is called *door*. The cells of the (input) tape can be imagined as an infinite series of rooms. We can say, that the head can only move through the doors.

Proof: Notions

The common boundary of two adjacent cells on the input tape is called *door*. The cells of the (input) tape can be imagined as an infinite series of rooms. We can say, that the head can only move through the doors.



The cells M_i and M_{i+1} and the door a between them.

A run of T

A run of T

Consider the truncated run of the Turing machine T on ω input:

$$\kappa_0(\omega) \to \kappa_1 \to \kappa_2 \to \ldots \to \kappa_\ell,$$

where

$$\ell := \min\{n \mid \text{ state of } \kappa_n \text{ is ACCEPT or REJECT.}\}$$

A run of T

Consider the truncated run of the Turing machine T on ω input:

$$\kappa_0(\omega) \to \kappa_1 \to \kappa_2 \to \ldots \to \kappa_\ell,$$

where

$$\ell := \min\{n \mid \text{ state of } \kappa_n \text{ is ACCEPT or REJECT.}\}$$

Definition: $\sigma(a, \omega)$

Now take those κ_j, κ_{j+1} configurations, in which the input eye is over M_i/M_{i+1}. Let s_j be the state of the Turing machine when passing through the a door separating the cells.
The sequence of these s_j states is denoted by $\sigma(a, \omega)$.

A crucial observation

A crucial observation

Let $\omega\overset{a}{|}$ be the initial segment of the input till the "door a", and let $\overset{a}{|}\omega$ be the final segment of the input after "door a".

A crucial observation

Let $\omega \overset{a}{|}$ be the initial segment of the input till the "door a", and let $\overset{a}{|}\omega$ be the final segment of the input after "door a".

Of course the concatenation of the two parts is the full ω input: $\omega = \left(\omega \overset{a}{|}\right)\left(\overset{a}{|}\omega\right)$.

A crucial observation

Let $\omega\overset{a}{|}$ be the initial segment of the input till the "door a", and let $\overset{a}{|}\omega$ be the final segment of the input after "door a".

Of course the concatenation of the two parts is the full ω input: $\omega = \left(\omega\overset{a}{|}\right)\left(\overset{a}{|}\omega\right)$.

Observation

Knowing $\omega\overset{a}{|}$ and the $\sigma(a,\omega)$ state sequence, then we are able to reconstruct how the Turing machine "works" when the eye is on the left hand side of the door a.

A crucial observation

Let $\omega \overset{a}{|}$ be the initial segment of the input till the "door a", and let $\overset{a}{|}\omega$ be the final segment of the input after "door a".

Of course the concatenation of the two parts is the full ω input: $\omega = \left(\omega \overset{a}{|}\right)\left(\overset{a}{|}\omega\right)$.

Observation

Knowing $\omega \overset{a}{|}$ and the $\sigma(a, \omega)$ state sequence, then we are able to reconstruct how the Turing machine "works" when the eye is on the left hand side of the door a.

We cannot know how long the eye was a right but as soon as it crosses the door (and as long as it remains on the left) we are able to describe the run of T.

Corollary of the Observation

Corollary of the Observation

Corollary

Let $\omega, \omega' \in \Sigma^n$ be arbitrary inputs and a be a a door. Suppose $\sigma(a, \omega) = \sigma(a, \omega')$ and Turing machine T has the same result on the two inputs.

Then T outputs the same on the input $\widetilde{\omega} = \left(\omega \overset{a}{|}\right)\left(\overset{a}{|} \omega'\right)$.

I_0

I_0

Definition: I_0

Suppose that $3|n$. Let

$$I_0 := \{\alpha 0^{\frac{n}{3}} \overleftarrow{\alpha} : \alpha \in \Sigma^{\frac{n}{3}}\} \subseteq \text{PALINDROM} \cap \Sigma^n.$$

I_0

Definition: I_0

Suppose that $3|n$. Let

$$I_0 := \{\alpha 0^{\frac{n}{3}} \overleftarrow{\alpha} : \alpha \in \Sigma^{\frac{n}{3}}\} \subseteq \text{PALINDROM} \cap \Sigma^n.$$

$$|I_0| = |\Sigma|^{\frac{n}{3}} = |\{0,1\}|^{\frac{n}{3}} = 2^{\frac{n}{3}}.$$

I_0

Definition: I_0

Suppose that $3|n$. Let

$$I_0 := \{\alpha 0^{\frac{n}{3}} \overleftarrow{\alpha} : \alpha \in \Sigma^{\frac{n}{3}}\} \subseteq \text{PALINDROM} \cap \Sigma^n.$$

$$|I_0| = |\Sigma|^{\frac{n}{3}} = |\{0,1\}|^{\frac{n}{3}} = 2^{\frac{n}{3}}.$$

Corollary

Let ω, $\omega' \in I_0$ be distinct words and a be a middle door (i.e. one of the doors separating the middle $n/3$ zeros). Then $\sigma(a, \omega) \neq \sigma(a, \omega')$.

I_0

Definition: I_0

Suppose that $3|n$. Let

$$I_0 := \{\alpha 0^{\frac{n}{3}} \overleftarrow{\alpha} : \alpha \in \Sigma^{\frac{n}{3}}\} \subseteq \text{PALINDROM} \cap \Sigma^n.$$

$$|I_0| = |\Sigma|^{\frac{n}{3}} = |\{0,1\}|^{\frac{n}{3}} = 2^{\frac{n}{3}}.$$

Corollary

Let ω, $\omega' \in I_0$ be distinct words and a be a middle door (i.e. one of the doors separating the middle $n/3$ zeros). Then $\sigma(a,\omega) \neq \sigma(a,\omega')$.

Indirectly, suppose that $\sigma(a,\omega) = \sigma(a,\omega')$.

I_0

Definition: I_0

Suppose that $3|n$. Let

$$I_0 := \{\alpha 0^{\frac{n}{3}} \overleftarrow{\alpha} : \alpha \in \Sigma^{\frac{n}{3}}\} \subseteq \text{PALINDROM} \cap \Sigma^n.$$

$$|I_0| = |\Sigma|^{\frac{n}{3}} = |\{0,1\}|^{\frac{n}{3}} = 2^{\frac{n}{3}}.$$

Corollary

Let ω, $\omega' \in I_0$ be distinct words and a be a middle door (i.e. one of the doors separating the middle $n/3$ zeros). Then $\sigma(a,\omega) \neq \sigma(a,\omega')$.

Indirectly, suppose that $\sigma(a,\omega) = \sigma(a,\omega')$. Then the previous corollary if both inputs have *ACCEPT* as the state of Turing machine, then the $\widetilde{\omega} = \left(\omega \overset{a}{|}\right)\left(\overset{a}{|} \omega'\right) = \alpha 0^{\frac{n}{3}} \overleftarrow{\alpha'}$ input is also accepted.

I_0

Definition: I_0

Suppose that $3|n$. Let

$$I_0 := \{\alpha 0^{\frac{n}{3}} \overleftarrow{\alpha} : \alpha \in \Sigma^{\frac{n}{3}}\} \subseteq \text{PALINDROM} \cap \Sigma^n.$$

$$|I_0| = |\Sigma|^{\frac{n}{3}} = |\{0,1\}|^{\frac{n}{3}} = 2^{\frac{n}{3}}.$$

Corollary

Let ω, $\omega' \in I_0$ be distinct words and a be a middle door (i.e. one of the doors separating the middle $n/3$ zeros). Then $\sigma(a,\omega) \neq \sigma(a,\omega')$.

Indirectly, suppose that $\sigma(a,\omega) = \sigma(a,\omega')$. Then the previous corollary if both inputs have *ACCEPT* as the state of Turing machine, then the $\widetilde{\omega} = \left(\omega \overset{a}{|}\right)\left(\overset{a}{|} \omega'\right) = \alpha 0^{\frac{n}{3}} \overleftarrow{\alpha'}$ input is also accepted. Contradiction.

A Lemma

A Lemma

Observation

The number of state sequences shorter than t

$$1 + |S| + \cdots + |S|^{t-1} = \frac{|S|^t - 1}{|S| - 1} < |S|^t - 1 < |S|^t.$$

A Lemma

Observation

The number of state sequences shorter than t

$$1 + |S| + \cdots + |S|^{t-1} = \frac{|S|^t - 1}{|S| - 1} < |S|^t - 1 < |S|^t.$$

Lemma

If $|I_0| = |\Sigma|^{\frac{n}{3}} \geq |S|^t$, then $\exists \omega \in I_0$ such that $\sigma(a, \omega)$ has length at least t, where a is a middle door.

A Lemma

Observation

The number of state sequences shorter than t

$$1 + |S| + \cdots + |S|^{t-1} = \frac{|S|^t - 1}{|S| - 1} < |S|^t - 1 < |S|^t.$$

Lemma

If $|I_0| = |\Sigma|^{\frac{n}{3}} \geq |S|^t$, then $\exists \omega \in I_0$ such that $\sigma(a, \omega)$ has length at least t, where a is a middle door.

We have $|I_0| = |\Sigma|^{\frac{n}{3}} \geq |S|^t$ in the case when $t \sim \beta_T \cdot n$.

A Corollary of the Lemma

A Corollary of the Lemma

Corollary

Let a be an arbitrary middle door. Assume that $|I_0| \geq 2|S|^t$. There exists at least $|I_0|/2$ inputs in I_0 for which $|\sigma(a, \omega)| \geq t$.

A Corollary of the Lemma

Corollary

Let a be an arbitrary middle door. Assume that $|I_0| \geq 2|S|^t$. There exists at least $|I_0|/2$ inputs in I_0 for which $|\sigma(a, \omega)| \geq t$.

Let denote the set of inputs in the Corollary by $I_1(a)$, i.e.

$$I_1(a) = \{\omega \in I_0 : |\sigma(a, \omega)| \geq t\} \subseteq I_0.$$

We also know that $t \sim \gamma_T \cdot n$ is a suitable choice to guarantee $|I_1| \geq |I_0|/2$.

The proof

The proof

$$\sum_{\omega \in I_0} TIME(\omega, T) \geq \sum_{\omega \in I_0} \sum_{a \text{ middle door}} |\{t : t \text{ the head crosses } a\}|$$

The proof

$$\sum_{\omega \in I_0} TIME(\omega, T) \geq \sum_{\omega \in I_0} \sum_{a \text{ middle door}} |\{t : t \text{ the head crosses } a\}|$$

$$= \sum_{\omega \in I_0} \sum_{a \text{ middle door}} |\sigma(a, \omega)| = \sum_{a \text{ middle door}} \sum_{\omega \in I_0} |\sigma(a, \omega)|$$

The proof

$$\sum_{\omega\in I_0} TIME(\omega, T) \geq \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\{t : t \text{ the head crosses } a\}|$$

$$= \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\sigma(a,\omega)| = \sum_{a \text{ middle door}} \sum_{\omega\in I_0} |\sigma(a,\omega)|$$

$$\geq \sum_{a \text{ middle door}} \sum_{\omega\in I_1(a)} t = \sum_{a \text{ middle door}} |I_1(a)| \cdot t \geq \sum_{a \text{ middle door}} \frac{|I_0|}{2} \cdot t$$

The proof

$$\sum_{\omega\in I_0} TIME(\omega, T) \geq \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\{t : t \text{ the head crosses } a\}|$$

$$= \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\sigma(a,\omega)| = \sum_{a \text{ middle door}} \sum_{\omega\in I_0} |\sigma(a,\omega)|$$

$$\geq \sum_{a \text{ middle door}} \sum_{\omega\in I_1(a)} t = \sum_{a \text{ middle door}} |I_1(a)| \cdot t \geq \sum_{a \text{ middle door}} \frac{|I_0|}{2} \cdot t$$

$$= \frac{n}{3} \cdot \frac{|I_0|}{2} \cdot t = \frac{\gamma_T}{6} \cdot n^2 \cdot |I_0|.$$

The proof

$$\sum_{\omega\in I_0} TIME(\omega, T) \geq \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\{t : t \text{ the head crosses } a\}|$$

$$= \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\sigma(a,\omega)| = \sum_{a \text{ middle door}} \sum_{\omega\in I_0} |\sigma(a,\omega)|$$

$$\geq \sum_{a \text{ middle door}} \sum_{\omega\in I_1(a)} t = \sum_{a \text{ middle door}} |I_1(a)| \cdot t \geq \sum_{a \text{ middle door}} \frac{|I_0|}{2} \cdot t$$

$$= \frac{n}{3} \cdot \frac{|I_0|}{2} \cdot t = \frac{\gamma_T}{6} \cdot n^2 \cdot |I_0|.$$

After division by $|I_0|$:

The proof

$$\sum_{\omega\in I_0} TIME(\omega, T) \geq \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\{t : t \text{ the head crosses } a\}|$$

$$= \sum_{\omega\in I_0} \sum_{a \text{ middle door}} |\sigma(a,\omega)| = \sum_{a \text{ middle door}} \sum_{\omega\in I_0} |\sigma(a,\omega)|$$

$$\geq \sum_{a \text{ middle door}} \sum_{\omega\in I_1(a)} t = \sum_{a \text{ middle door}} |I_1(a)| \cdot t \geq \sum_{a \text{ middle door}} \frac{|I_0|}{2} \cdot t$$

$$= \frac{n}{3} \cdot \frac{|I_0|}{2} \cdot t = \frac{\gamma_T}{6} \cdot n^2 \cdot |I_0|.$$

After division by $|I_0|$:

$$\frac{1}{|I_0|} \sum_{\omega\in I_0} TIME(\omega, T) \geq \frac{\gamma_T}{6} \cdot n^2.$$

This is the end!

Thank you for your attention!