Computability

Key question

Are there problems that cannot be solved by a computer?
(i.e. given a problem, can we always design an *algorithm* to solve it?)

Key question

Are there problems that cannot be solved by a computer?
(i.e. given a problem, can we always design an *algorithm* to solve it?)

David Hibert's 1900 problem:
Given a polynomial on multiple variables (e.g. $4xy^2 - 5x^3 + 1$),
test if the polynomial has an "integral" root.

Definition of algorithm

Algorithm (or procedure/recipe): a collection of steps to carry out some computation task

Want: a mathematical definition for algorithm

Definition of algorithm

Algorithm (or procedure/recipe): a collection of steps to carry out some computation task

Want: a mathematical definition for algorithm

1936: Alan Turing defined an algorithm as a Turing-machine

1936: Alonzo Church defined an algorithm as λ-calculus

These two definitions turn out equivalent:
"Church-Turing thesis"

Algorithm

Algorithm (intuition)	$\Leftrightarrow$	Turing machine (math)

Does there exist an algorithm that solves a problem?

Problem $P \longrightarrow$ language L

Algorithm solves $P \longrightarrow$ TM decides L

(not enough: TM recognizes L)

Algorithm

Algorithm (intuition) $\Leftrightarrow$ Turing machine (math)

Does there exist an algorithm that solves a problem?

Problem $P \longrightarrow$ language L

Algorithm solves $P \longrightarrow$ TM decides L

(not enough: TM recognizes L)

This is the topic of "computability":

focus on computation models

$\longrightarrow$ focus on using TM to study algorithms

From now on, we use "high-level description" of TM

Example of high-level TM

Consider $L = \{w\#w \mid w \in \{0,1\}^*\}$

The following Turing machine decides L:

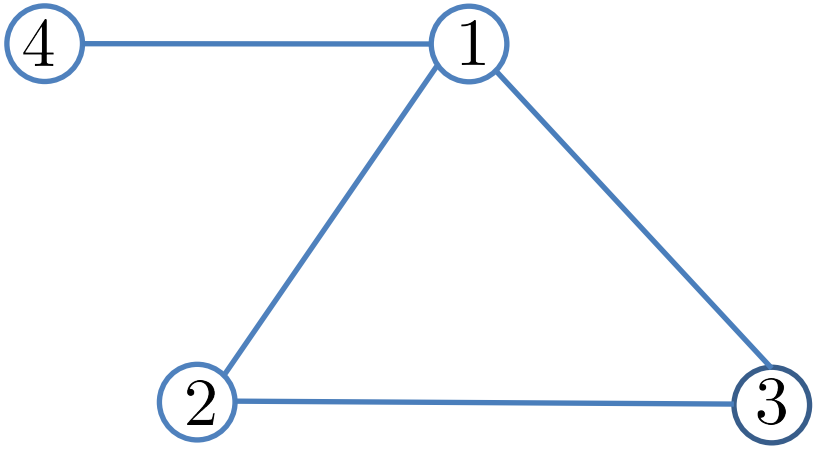
$M =$ "On input string s:

1) Zig-zag across tape to corresponding positions on both sides of #, and check if these positions contain the same symbols. If they do not or if no # found, then reject. Cross off symbols that are checked.

2) When all symbols to the left of # are crossed off, check for any remaining symbols to the right of #. If any symbols remain, then reject.
Otherwise, accept."

Example of high-level TM

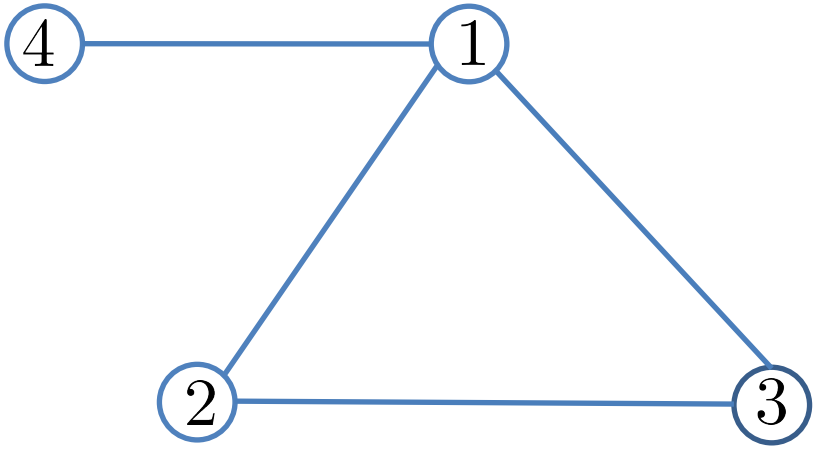
An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set $\mathcal{V}$ of *nodes* and a set of $\mathcal{E}$ of *edges*



$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$
$$\mathcal{V} = \{0, 1, 2, 3\}$$
$$\mathcal{E} = \{(1, 2), (2, 3), (3, 1), (1, 4)\}$$

Example of high-level TM

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set $\mathcal{V}$ of *nodes* and a set of $\mathcal{E}$ of *edges*



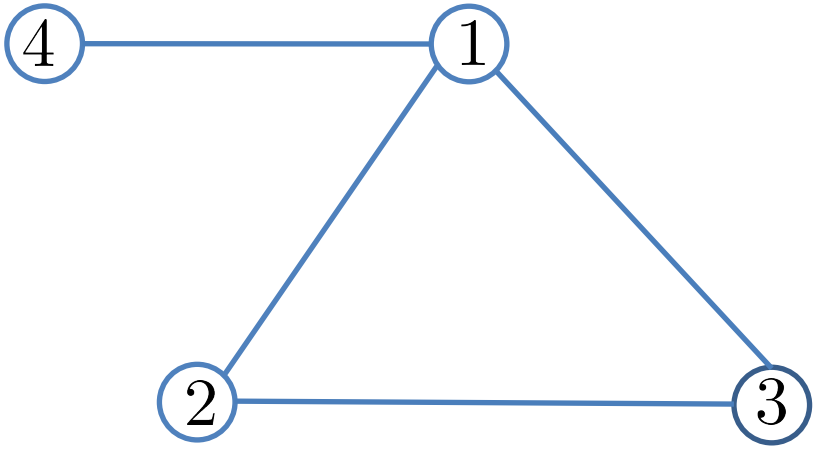
$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$
$$\mathcal{V} = \{0, 1, 2, 3\}$$
$$\mathcal{E} = \{(1, 2), (2, 3), (3, 1), (1, 4)\}$$

Write $< \mathcal{G} >$ for the string that encodes graph $\mathcal{G}$

e.g. $< \mathcal{G} > = (1, 2, 3, 4)((1, 2), (2, 3), (3, 1), (1, 4))$

Example of high-level TM

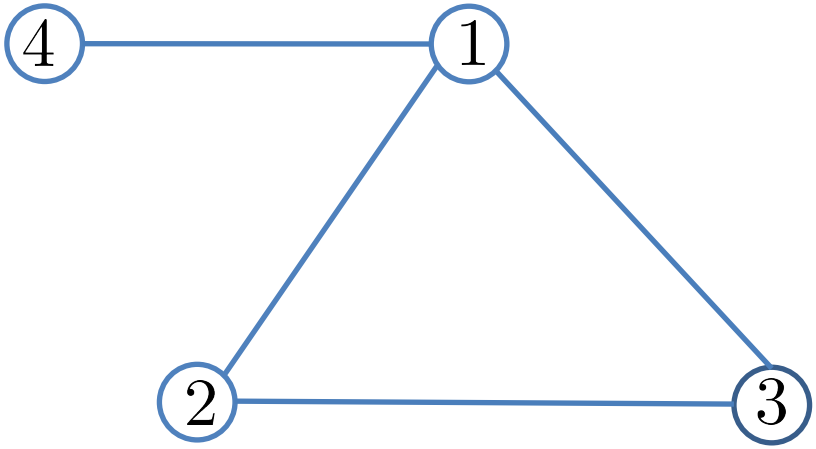
$\mathcal{G}$ is *connected* if every node can reach every other node by a sequence of edges



$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$
$$\mathcal{V} = \{0, 1, 2, 3\}$$
$$\mathcal{E} = \{(1, 2), (2, 3), (3, 1), (1, 4)\}$$

Example of high-level TM

$\mathcal{G}$ is *connected* if every node can reach every other node by a sequence of edges



$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$
$$\mathcal{V} = \{0, 1, 2, 3\}$$
$$\mathcal{E} = \{(1, 2), (2, 3), (3, 1), (1, 4)\}$$

Let L be the language consisting of all strings representing undirected connected graphs

Then $L := \{< \mathcal{G} > | \ \mathcal{G}$ is an undirected connected graph$\}$

Example of high-level TM

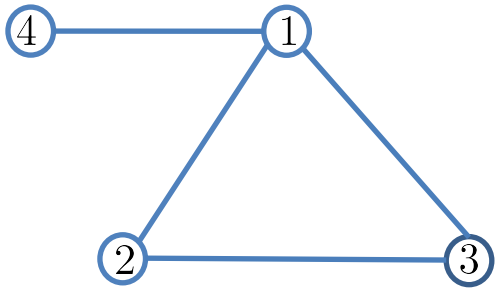
Consider $L := \{< \mathcal{G} > | \, \mathcal{G} \text{ is an undirected connected graph}\}$

Design a TM that decides L

Example of high-level TM

Consider $L := \{< \mathcal{G} > | \, \mathcal{G}$ is an undirected connected graph$\}$

Design a TM that decides L



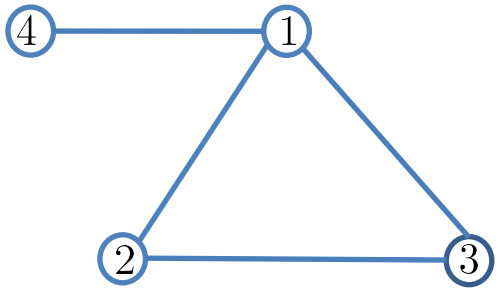
M = "On input string $< \mathcal{G} >$:

1) Select the first node and mark it.

Example of high-level TM

Consider $L := \{< \mathcal{G} > | \, \mathcal{G}$ is an undirected connected graph$\}$

Design a TM that decides L



$\mathbf{M}$ = "On input string $< \mathcal{G} >$:

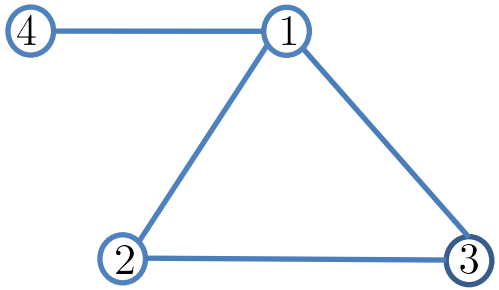
1) Select the first node and mark it.

2) From a marked node v, select an unmarked node v' that is connected to v and mark v'.
Repeat this step until no unmarked nodes can be selected.

Example of high-level TM

Consider $L := \{< \mathcal{G} > | \, \mathcal{G}$ is an undirected connected graph$\}$

Design a TM that decides L



M = "On input string $< \mathcal{G} >$:

1) Select the first node and mark it.

2) From a marked node v, select an unmarked node v' that is connected to v and mark v'.
Repeat this step until no unmarked nodes can be selected.

3) Scan all nodes to determine if they are all marked.
If so, accept. Otherwise, reject."

Example of high-level TM

So $L := \{< \mathcal{G} > | \ \mathcal{G}$ is an undirected connected graph$\}$
is a Turing-decidable language

Problem: given an undirected graph, test if the graph is connected.

Problem $P \longrightarrow$ language L

Algorithm solves $P \longrightarrow$ TM decides L

Example of high-level TM

Are there any problems that are not algorithmically solvable?

David Hibert's 1900 problem:
Given a polynomial on multiple variables (e.g. $4xy^2 - 5x^3 + 1$),
test if the polynomial has an "integral" root.

Example of high-level TM

Are there any problems that are not algorithmically solvable?

David Hibert's 1900 problem:
Given a polynomial on multiple variables (e.g. $4xy^2 - 5x^3 + 1$), test if the polynomial has an "integral" root.

$< p >= (4xyy - 5xxx + 1)$

$L := \{< p > |\ p \text{ has an integral root}\}$

$\mathbf{M}$ = "On input string $< p >$:

1) Evaluate p with x, y set respectively to integers $0, 1, -1, 2, -2, \ldots$. If p is ever evaluated to 0, accept."

Note $\mathbf{M}$ only recognizes L, but does not decide L

Decidability

Decidability & undecidability

Objective: explore the limits of algorithmic solvability

and identify problems that can be solved algorithmically, while others cannot be

Decidability & undecidability

Objective: explore the limits of algorithmic solvability

and identify problems that can be solved algorithmically, while others cannot be

Why study "unsolvability"?

Useful in the sense: you know the problem must be simplified or altered before a solution can be found.
You don't need to waste more time on the original problem.

Computer is a tool after all.
Like any tool, computer has limitations.

Decidable problems

Acceptance problem for DFA:

Given a DFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$ and a string $s \in \Sigma^*$, test if $\mathbf{G}$ accepts s

Strategy: represent the problem by a language, and show the language is Turing-decidable

Decidable problems

Acceptance problem for DFA:

Given a DFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$ and a string $s \in \Sigma^*$, test if $\mathbf{G}$ accepts s

Strategy: represent the problem by a language, and show the language is Turing-decidable

Consider $A_{\text{DFA}} = \{< \mathbf{G}, s > |\ s \in L_a(\mathbf{G})\}$

$< \mathbf{G}, s >$ is the string that encodes DFA $\mathbf{G}$ and string s

Design a TM that decides A_{DFA}

Decidable problems

Acceptance problem for DFA:

Given a DFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$ and a string $s \in \Sigma^*$, test if $\mathbf{G}$ accepts s

$\mathbf{M}_{\text{DFA}} =$ "On input string $< \mathbf{G}, s >$:

1) Run s on $\mathbf{G}$ from the initial state q_0

2) If the run ends at an accept state in Q_a, accept. Otherwise, reject."

Decidable problems

Acceptance problem for NFA:

Given a NFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$ and a string $s \in \Sigma^*$, test if $\mathbf{G}$ accepts s

Decidable problems

Acceptance problem for NFA:

Given a NFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$ and a string $s \in \Sigma^*$, test if $\mathbf{G}$ accepts s

Consider $A_{\text{NFA}} = \{< \mathbf{G}, s > |\ s \in L_a(\mathbf{G})\}$

$\mathbf{M}_{\text{NFA}} =$ "On input string $< \mathbf{G}, s >$:

1) Convert $\mathbf{G}$ to an equivalent DFA $\mathbf{G}'$
2) Run TM (decider) $\mathbf{M}_{\text{DFA}}$ on input $< \mathbf{G}', s >$
3) If $\mathbf{M}_{\text{DFA}}$ accepts, accept. Otherwise, reject."

($\mathbf{M}_{\text{NFA}}$ uses $\mathbf{M}_{\text{DFA}}$ as a *subprocedure*)

Decidable problems

Acceptance problem for regular expression:

Given a regular expression R and a string $s \in \Sigma^*$, test if R generates s

Consider $A_{\mathrm{RE}} = \{< R, s > |\ s \in L(R)\}$

$\mathbf{M}_{\mathrm{RE}}$ = "On input string $< R, s >$:

1) Convert R to an equivalent NFA $\mathbf{G}$
2) Run TM (decider) $\mathbf{M}_{\mathrm{NFA}}$ on input $< \mathbf{G}, s >$
3) If $\mathbf{M}_{\mathrm{NFA}}$ accepts, accept. Otherwise, reject."

($\mathbf{M}_{\mathrm{RE}}$ uses $\mathbf{M}_{\mathrm{NFA}}$ as a *subprocedure*)

Decidable problems

Emptyness testing problem for DFA:

Given a DFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$,
test if $\mathbf{G}$ accepts no string, i.e. $L_a(\mathbf{G}) = \emptyset$

Decidable problems

Emptyness testing problem for DFA:

Given a DFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$,
test if $\mathbf{G}$ accepts no string, i.e. $L_a(\mathbf{G}) = \emptyset$

Consider $E_{\mathrm{DFA}} = \{< \mathbf{G} > | \; L_a(\mathbf{G}) = \emptyset\}$

Design a TM that decides E_{DFA}

Decidable problems

Emptyness testing problem for DFA:

Given a DFA $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$,
test if $\mathbf{G}$ accepts no string, i.e. $L_a(\mathbf{G}) = \emptyset$

Consider $E_{\text{DFA}} = \{< \mathbf{G} > | \; L_a(\mathbf{G}) = \emptyset\}$

Design a TM that decides E_{DFA}

$\mathbf{M}_{\text{DFA}}$ = "On input string $< \mathbf{G} >$:

1) Mark the initial state q_0

2) From a marked state q, select an unmarked state q' s.t. $q' = \delta(q, \sigma)$ for some σ and mark q'.
Repeat this step until no unmarked states can be selected.

3) If no accept state is marked, accept. Otherwise, reject."

Recap

Acceptance problem for DFA (NFA, regular expression)
is solvable

Emptyness testing problem for DFA (NFA, regular expression)
is solvable

Recap

Acceptance problem for DFA (NFA, regular expression)
is solvable

Emptyness testing problem for DFA (NFA, regular expression)
is solvable

Fact:
Every regular language is decidable
Every context-free language is decidable