CS 301

Lecture 23 – Time complexity

Stephen Checkoway

April 23, 2018

Complexity

Computability What languages are decidable? (Equivalently, what decision problems can we solve with a computer?)

Complexity

Computability What languages are decidable? (Equivalently, what decision problems can we solve with a computer?)

Complexity How long does it take to check if a string is in a decidable language? (Equivalently, how long does it take to answer a decision question about an instance of a problem?)

Running time

The running time of a decider M is a function $t : \mathbb{N} \to \mathbb{N}$ where $t(n)$ is the maximum number of steps M takes to accept/reject any string of length n

This is the worst-case time: If M can accept/reject every string of length 5 except aabaa in 15 steps, but aabaa takes 4087 steps, then $t(5) = 4087$

Big-O review

If $f, g : \mathbb{N} \to \mathbb{R}^+$, we say $f(n) = O(g(n))$ to mean there exist $N, c > 0$ such that for all $n \geq N$, $f(n) \leq c \cdot g(n)$

Examples

Constant $c = O(1)$ for any $c \in \mathbb{R}^+$

Big-O review

If $f, g : \mathbb{N} \to \mathbb{R}^+$, we say $f(n) = O(g(n))$ to mean there exist $N, c > 0$ such that for all $n \geq N$, $f(n) \leq c \cdot g(n)$

Examples

Constant $c = O(1)$ for any $c \in \mathbb{R}^+$

Polynomial $a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 = O(n^k)$

Big-O review

If $f, g : \mathbb{N} \to \mathbb{R}^+$, we say $f(n) = O(g(n))$ to mean there exist $N, c > 0$ such that for all $n \geq N$, $f(n) \leq c \cdot g(n)$

Examples

Constant	$c = O(1)$ for any $c \in \mathbb{R}^+$
Polynomial	$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 = O(n^k)$
Logarithmic	$a \log_b n = O(\log n)$

Big-O review

If $f, g : \mathbb{N} \to \mathbb{R}^+$, we say $f(n) = O(g(n))$ to mean there exist $N, c > 0$ such that for all $n \geq N$, $f(n) \leq c \cdot g(n)$

Examples

Constant	$c = O(1)$ for any $c \in \mathbb{R}^+$
Polynomial	$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 = O(n^k)$
Logarithmic	$a \log_b n = O(\log n)$
Arithmetic	$O(n^2) + O(n \log^2 n \cdot \log \log n) = O(n^2)$

Big-O review

If $f, g : \mathbb{N} \to \mathbb{R}^+$, we say $f(n) = O(g(n))$ to mean there exist $N, c > 0$ such that for all $n \geq N$, $f(n) \leq c \cdot g(n)$

Examples

Constant	$c = O(1)$ for any $c \in \mathbb{R}^+$
Polynomial	$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 = O(n^k)$
Logarithmic	$a \log_b n = O(\log n)$
Arithmetic	$O(n^2) + O(n \log^2 n \cdot \log \log n) = O(n^2)$
Polynomial bound	$2^{O(\log n)}$ or $n^{O(1)}$

Big-O review

If $f, g : \mathbb{N} \to \mathbb{R}^+$, we say $f(n) = O(g(n))$ to mean there exist $N, c > 0$ such that for all $n \geq N$, $f(n) \leq c \cdot g(n)$

Examples

Constant	$c = O(1)$ for any $c \in \mathbb{R}^+$
Polynomial	$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 = O(n^k)$
Logarithmic	$a \log_b n = O(\log n)$
Arithmetic	$O(n^2) + O(n \log^2 n \cdot \log \log n) = O(n^2)$
Polynomial bound	$2^{O(\log n)}$ or $n^{O(1)}$
Exponential bound	$2^{O(n^\delta)}$ for $\delta > 0$

Little-O review

If $f, g : \mathbb{N} \to \mathbb{R}^+$, we say $f(n) = o(g(n))$ to mean

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

Equivalently, there exist $N, c > 0$ such that for all $n \geq N$, $f(n) < c \cdot g(n)$

Analyzing running time of deciders

It's too much work to be precise (we don't want to think about states)

For implementation-level descriptions of TMs, we can use big-O to describe the running time

Example

Consider the TM M_1 which decides $A = \{0^n 1^n \mid n \geq 0\}$

M_1 = "On input 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 any 0 or 1 remain uncrossed off, then *reject*; otherwise *accept*"

How long does M_1 take to accept/reject a string of length n?

Example

Consider the TM M_1 which decides $A = \{0^n1^n \mid n \geq 0\}$

M_1 = "On input 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 any 0 or 1 remain uncrossed off, then *reject*; otherwise *accept*"

How long does M_1 take to accept/reject a string of length n?

Analyze each step

Example

Consider the TM M_1 which decides $A = \{0^n1^n \mid n \geq 0\}$

M_1 = "On input 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 any 0 or 1 remain uncrossed off, then *reject*; otherwise *accept*"

How long does M_1 take to accept/reject a string of length n?

Analyze each step

1. Scanning across the tape takes $O(n)$

Example

Consider the TM M_1 which decides $A = \{0^n 1^n \mid n \geq 0\}$

M_1 = "On input 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 any 0 or 1 remain uncrossed off, then *reject*; otherwise *accept*"

How long does M_1 take to accept/reject a string of length n?

Analyze each step

1. Scanning across the tape takes $O(n)$
2. Checking if 0 or 1 remain takes $O(n)$

Example

Consider the TM M_1 which decides $A = \{0^n 1^n \mid n \geq 0\}$

M_1 = "On input 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 any 0 or 1 remain uncrossed off, then *reject*; otherwise *accept*"

How long does M_1 take to accept/reject a string of length n?

Analyze each step

1. Scanning across the tape takes $O(n)$
2. Checking if 0 or 1 remain takes $O(n)$
3. Crossing off one 0 and one 1 takes $O(n)$

Example

Consider the TM M_1 which decides $A = \{0^n 1^n \mid n \geq 0\}$

M_1 = "On input 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 any 0 or 1 remain uncrossed off, then *reject*; otherwise *accept*"

How long does M_1 take to accept/reject a string of length n?

Analyze each step

1. Scanning across the tape takes $O(n)$
2. Checking if 0 or 1 remain takes $O(n)$
3. Crossing off one 0 and one 1 takes $O(n)$
4. Performing the final check takes $O(n)$

Example

Consider the TM M_1 which decides $A = \{0^n 1^n \mid n \geq 0\}$

M_1 = "On input 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 any 0 or 1 remain uncrossed off, then *reject*; otherwise *accept*"

How long does M_1 take to accept/reject a string of length n?

Analyze each step

1. Scanning across the tape takes $O(n)$
2. Checking if 0 or 1 remain takes $O(n)$
3. Crossing off one 0 and one 1 takes $O(n)$
4. Performing the final check takes $O(n)$

Each time through the loop takes $O(n) + O(n) = O(n)$ time and the loop happens at most $n/2$ times

The total running time is $O(n) + (n/2)O(n) + O(n) = O(n^2)$

Time complexity class

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The time complexity class $\mathrm{TIME}(t(n))$ is the set of languages that are decidable by an $O(t(n))$-time TM

Example

$A = \{0^n 1^n \mid n \geq 0\} \in \mathrm{TIME}(n^2)$ because we gave a TM M_1 that decides A in $O(n^2)$ time

Time complexity class

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The time complexity class $\mathrm{TIME}(t(n))$ is the set of languages that are decidable by an $O(t(n))$-time TM

Example

$A = \{0^n 1^n \mid n \geq 0\} \in \mathrm{TIME}(n^2)$ because we gave a TM M_1 that decides A in $O(n^2)$ time

Sipser gives a more clever TM M_2 that decides A in time $O(n \log n)$ by crossing off every other 0 and every other 1 each time through the loop

Thus, $A \in \mathrm{TIME}(n \log n)$ (this is the best we can do on a single-tape TM)

What about a 2-TM?

With a 2-TM, we can decide A in linear ($O(n)$) time

M_3 = "On input w,

1. Scan right and *reject* if any 0 follows a 1
2. Return the beginning of the first tape
3. Scan right to the first 1, copying the 0s to the second tape
4. Scan right on the first tape and left on the second, crossing off a 0 for each 1, if there aren't enough 0s, then *reject*
5. If more 0s remain, then *reject*; otherwise *accept*"

Steps 1 and 2 each take $O(n)$; together, steps 3, 4, and 5 constitute a single pass over the input so $O(n)$

Total running time: $O(n) + O(n) + O(n) = O(n)$

Time complexity of a language depends on our model of computation

M_1 decides A in time $O(n^2)$

M_2 decides A in time $O(n \log n)$

M_3 decides A in time $O(n)$ but uses a 2-TM

Relationships between models of computation

Recall from computability that the following are equivalent

- Single tape TM
- k-tape TM
- Nondeterministic TM

The situation for complexity is different

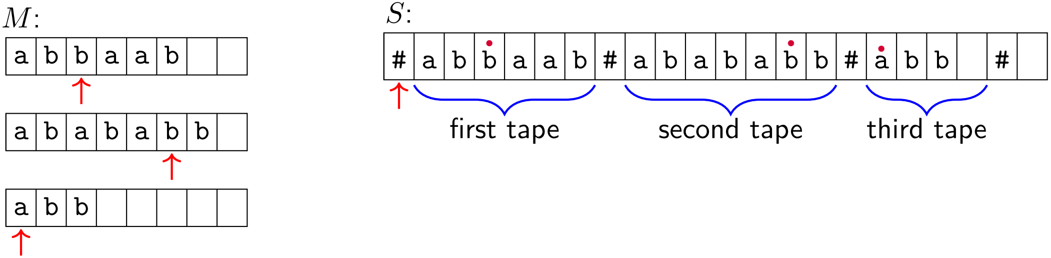
Simulating a k-TM

Theorem

Let $t : \mathbb{N} \to \mathbb{R}^+$ where $t(n) \geq n$. Every $t(n)$-time k-TM has an equivalent $O(t^2(n))$-time single-tape TM

Proof

Recall that we simulated a k-TM M with a single-tape TM S by writing the k tapes separated with $\#$ and dots representing the heads; e.g.,



Proof continued

If M runs in time $t(n)$, then it uses at most $t(n)$ tape cells on each tape so S will use at most $k \cdot t(n) + k + 1 = O(t(n))$ cells

Simulating one step of M required scanning across the tape twice and performing up to k shifts [why?]

Thus, each step of M takes $O(t(n))$ time for S to simulate

Since there are $t(n)$ steps and each takes $O(t(n))$ time, the running time for S is $t(n) \cdot O(t(n)) = O(t^2(n))$ $\square$

Simulating a k-TM with a 2-TM

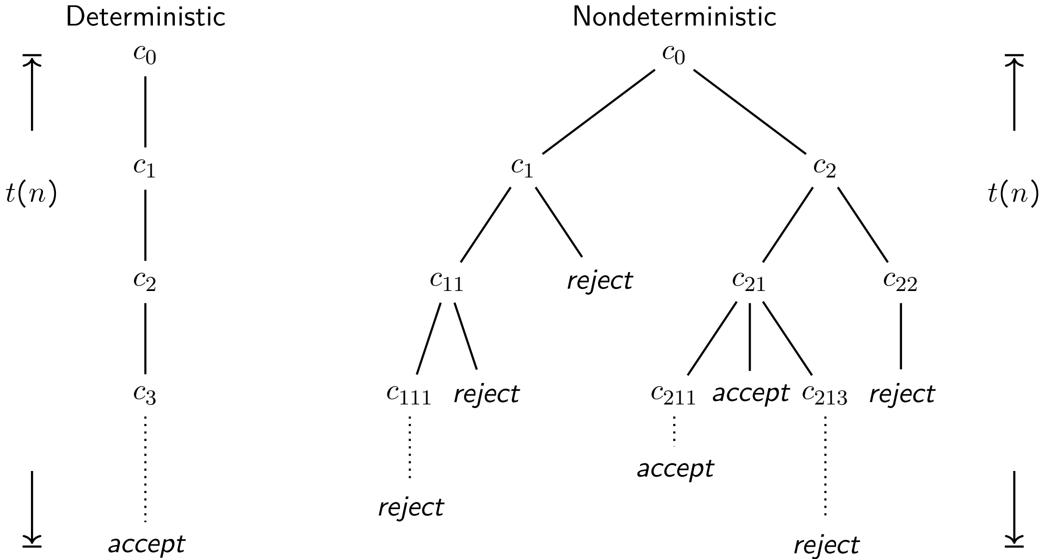
Just for your own edification:

Theorem

Every k tape TM that runs in time $t(n)$ for $t(n) \geq n$ can be simulated by a 2-tape TM in time $O(t(n) \log t(n))$

Running time for NTMs

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



Simulating an NTM

Theorem
Every $t(n)$-time NTM where $t(n) \geq n$ has an equivalent deterministic $2^{O(t(n))}$-time TM

Simulating an NTM

Theorem

Every $t(n)$-time NTM where $t(n) \geq n$ has an equivalent deterministic $2^{O(t(n))}$-time TM

Proof idea

Our simulation of an NTM used a 3-TM and it performed a breadth first search of the configuration tree

The height of the tree is $t(n)$ and if the maximum number of choices at each step is b, then the tree has $O(b^{t(n)})$ total nodes

Simulating an NTM

Theorem

Every $t(n)$-time NTM where $t(n) \geq n$ has an equivalent deterministic $2^{O(t(n))}$-time TM

Proof idea

Our simulation of an NTM used a 3-TM and it performed a breadth first search of the configuration tree

The height of the tree is $t(n)$ and if the maximum number of choices at each step is b, then the tree has $O(b^{t(n)})$ total nodes

For each node, we simulate from the root to the node which takes $O(t(n))$ time

The running time of the 3-TM is $O(t(n)) \cdot O(b^{t(n)}) = 2^{O(t(n))}$

Simulating an NTM

Theorem

Every $t(n)$-time NTM where $t(n) \geq n$ has an equivalent deterministic $2^{O(t(n))}$-time TM

Proof idea

Our simulation of an NTM used a 3-TM and it performed a breadth first search of the configuration tree

The height of the tree is $t(n)$ and if the maximum number of choices at each step is b, then the tree has $O(b^{t(n)})$ total nodes

For each node, we simulate from the root to the node which takes $O(t(n))$ time

The running time of the 3-TM is $O(t(n)) \cdot O(b^{t(n)}) = 2^{O(t(n))}$

We can simulate the 3-TM with a TM in time $\left(2^{O(t(n))}\right)^2 = 2^{O(t(n))}$

Polynomial time

Note that the time to decide a language with a TM takes only a polynomial (a square) of the time it takes to decide with a k-TM

All *reasonable* deterministic models of computation are polynomially equivalent; that is, you can simulate any of them with any other with only a polynomial slow down

As we saw, nondeterminism seems fundamentally different

From this point, we're not going to be concerned with polynomial differences in time; e.g., the difference between $O(n \log n)$ and $O(n^{105})$ won't matter: Both are $n^{O(1)}$

The class P

P is the class of languages that are decidable in polynomial time on a deterministic TM,

$$\mathrm{P} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(n^k)$$

P is a useful class because membership in P doesn't depend on (reasonable) deterministic models of computation

A problem that can be solved in polynomial time on a computer can be solved in polynomial time on a TM (even though the polynomial for one may be much larger than for the other)

The class $\mathrm{EXPTIME}$

$\mathrm{EXPTIME}$ is the class of languages that are decidable in exponential time on a deterministic TM

$$\mathrm{EXPTIME} = \bigcup_{k=0}^{\infty} \mathrm{TIME}(2^{n^k})$$

Note that $\mathrm{EXPTIME}$ is the same for any polynomially-equivalent models of computation

If language A takes time $2^{O(n^k)}$ under one model, then it'll take $\left(2^{O(n^k)}\right)^c = 2^{c \cdot O(n^k)} = 2^{O(n^k)}$ time under a polynomially-equivalent model

Tractable and intractable problems

We say that problems that can be solved in polynomial time are tractable: We can solve them with computers

We say that problems that take exponential time (or longer) are intractable: We can only solve very small instances of them with computers

P = tractable
$\mathrm{EXPTIME}$ = intractable

Lots of interesting problems are in P!

Graphs

Recall: A graph G is a pair $G = (V, E)$ where V is the set of vertices and $E \subseteq V \times V$ is the set of edges

- For an undirected graph edge $(a, b) = (b, a)$ (sometimes we write $\{a, b\}$)
- For a directed graph edge (a, b) is different from edge (b, a) (unless $a = b$)

In an algorithms class (e.g., CS 401), we would care about run times of algorithms in terms of $m = |V|$ and $n = |E|$

But since $n \leq m^2$ and we don't care about polynomial differences, we'll talk about graph algorithm run times in terms of m alone

That is, we're going to phrase problems involving graphs as languages (of course) and we're going to ask questions like is the language in P?

$\mathrm{PATH} \in \mathrm{P}$

Define $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a directed graph and there's a path from s to $t\}$.
Then $\mathrm{PATH} \in \mathrm{P}$

$\mathrm{PATH} \in \mathrm{P}$

Define $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a directed graph and there's a path from s to $t\}$.
Then $\mathrm{PATH} \in \mathrm{P}$

We can give a TM M to decide PATH
M = "On input $\langle G, s, t\rangle$ where $G = (V, E)$ and $s, t \in V$,

1. Mark s
2. Repeat until no new nodes are marked,
3. For each $(x, y) \in E$, if x is marked and y is not, mark y
4. If t is marked, then *accept*; otherwise *reject*"

PATH $\in$ P

Define $\text{PATH} = \{\langle G, s, t\rangle \mid G$ is a directed graph and there's a path from s to $t\}$.
Then $\text{PATH} \in \text{P}$

We can give a TM M to decide PATH
M = "On input $\langle G, s, t\rangle$ where $G = (V, E)$ and $s, t \in V$,

1. Mark s
2. Repeat until no new nodes are marked,
3. For each $(x, y) \in E$, if x is marked and y is not, mark y
4. If t is marked, then *accept*; otherwise *reject*"

The algorithm marks all nodes reachable from node s and accepts iff t is marked so $L(M) = \text{PATH}$.

PATH ∈ P

Define $\mathrm{PATH} = \{\langle G, s, t\rangle \mid G$ is a directed graph and there's a path from s to $t\}$.
Then $\mathrm{PATH} \in \mathrm{P}$

We can give a TM M to decide PATH
M = "On input $\langle G, s, t\rangle$ where $G = (V, E)$ and $s, t \in V$,

1. Mark s
2. Repeat until no new nodes are marked,
3.   For each $(x, y) \in E$, if x is marked and y is not, mark y
4. If t is marked, then *accept*; otherwise *reject*"

The algorithm marks all nodes reachable from node s and accepts iff t is marked so $L(M) = \mathrm{PATH}$.

The loop in step 2 happens at most $m = |V|$ times and there are at most $n = |E| \le m^2$ edges to check each time. Therefore, the running time is polynomial in m and thus polynomial in the size of the input

What about on a computer?

Implementing this algorithm on a computer would take $O(mn)$ time since it is looping over each of the n edges at most m times

There's a more clever algorithm that takes time $O(m + n)$ but since both of these are polynomials, we don't need to be any more clever

Boolean formulae

A boolean formula is an expression containing boolean variables and operations ($\wedge$, $\vee$, and $\neg$)

Example: $\phi = (\neg x \wedge y) \vee (x \wedge \neg z)$

As a shorthand, we write $\overline{x}$ for $\neg x$ so $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$

Boolean formulae

A boolean formula is an expression containing boolean variables and operations ($\wedge$, $\vee$, and $\neg$)

Example: $\phi = (\neg x \wedge y) \vee (x \wedge \neg z)$

As a shorthand, we write $\overline{x}$ for $\neg x$ so $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$

A boolean formula is in conjunctive normal form (CNF) if it consists of conjunctions (ANDs) of disjunctions (ORs)

- $(a \vee \overline{b} \vee \overline{c}) \wedge (\overline{d} \vee e \vee f)$
- $(a \vee b) \wedge c$
- $a \vee b$ [Why is this in CNF?]
- a

Terminology

Literal A variable or its negation: x, $\overline{y}$, z are all literals

Terminology

Literal A variable or its negation: x, $\overline{y}$, z are all literals

Clause A disjunction (OR) of literals: $x \vee y \vee \overline{z}$

Terminology

Literal A variable or its negation: x, $\overline{y}$, z are all literals

Clause A disjunction (OR) of literals: $x \vee y \vee \overline{z}$

k-CNF A formula in CNF where each clause contains exactly k literals
Example 2-CNF formula

$$\phi = \underbrace{(a \vee b)}_{\text{clause}} \wedge (\overline{a} \vee c) \wedge (\overline{b} \vee \overline{c})$$

Terminology

Literal A variable or its negation: x, $\overline{y}$, z are all literals

Clause A disjunction (OR) of literals: $x \vee y \vee \overline{z}$

k-CNF A formula in CNF where each clause contains exactly k literals
Example 2-CNF formula

$$\phi = \underbrace{(a \vee b)}_{\text{clause}} \wedge (\overline{a} \vee c) \wedge (\overline{b} \vee \overline{c})$$

Satisfiable A formula is satisfiable is there is an assignment of truth values (T/F or $1/0$) to the variables that makes the whole formula true
ϕ is satisfiable by setting $a = T$, $b = F$, and $c = T$

Terminology

Literal A variable or its negation: x, $\overline{y}$, z are all literals

Clause A disjunction (OR) of literals: $x \vee y \vee \overline{z}$

k-CNF A formula in CNF where each clause contains exactly k literals
Example 2-CNF formula

$$\phi = \underbrace{(a \vee b)}_{\text{clause}} \wedge (\overline{a} \vee c) \wedge (\overline{b} \vee \overline{c})$$

Satisfiable A formula is satisfiable is there is an assignment of truth values (T/F or $1/0$) to the variables that makes the whole formula true
ϕ is satisfiable by setting $a = T$, $b = F$, and $c = T$

Unsatisfiable A formula is unsatisfiable if every assignment of truth values to the variables makes the whole formula false
$\psi = (a \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b}) \wedge (a \vee b)$ is unsatisfiable because every assignment makes one of the four clauses false

2-SAT

Define 2-SAT = $\{\langle\phi\rangle \mid \phi$ is a satisfiable boolean formula in 2-CNF$\}$

2-SAT

Define $2\text{-SAT} = \{\langle\phi\rangle \mid \phi$ is a satisfiable boolean formula in 2-CNF$\}$

2-SAT is decidable
M_1 = "On input $\langle\phi\rangle$,

1. For each assignment of truth values to variables in ϕ,
2.   If the assignment satisfies ϕ, then *accept*
3. Otherwise, *reject*"

Clearly, M_1 decides 2-SAT. What is its run time?

2-SAT

Define $\text{2-SAT} = \{\langle\phi\rangle \mid \phi$ is a satisfiable boolean formula in 2-CNF$\}$

2-SAT is decidable
M_1 = "On input $\langle\phi\rangle$,

1. For each assignment of truth values to variables in ϕ,
2. If the assignment satisfies ϕ, then *accept*
3. Otherwise, *reject*"

Clearly, M_1 decides 2-SAT. What is its run time?

If there are n variables, then there are 2^n combinations of assignments to try so $\text{2-SAT} \in \text{EXPTIME}$. Can we do better?

Implications

Recall that the logical implication $a \to b$ is equivalent to $\overline{a} \vee b$

Thus $x \vee y$ is equivalent to $\overline{x} \to y$ and $\overline{y} \to x$

Implications

Recall that the logical implication $a \rightarrow b$ is equivalent to $\overline{a} \vee b$

Thus $x \vee y$ is equivalent to $\overline{x} \rightarrow y$ and $\overline{y} \rightarrow x$

From a formula in 2-CNF, we can produce a set of implications which are all simultaneously satisfiable if the formula is

$\phi = (a \vee b) \wedge (\overline{a} \vee c) \wedge (\overline{b} \vee \overline{c})$

$$\begin{array}{ll} \overline{a} \rightarrow b & \overline{b} \rightarrow a \\ a \rightarrow c & \overline{c} \rightarrow \overline{a} \\ b \rightarrow \overline{c} & c \rightarrow \overline{b} \end{array}$$

$\psi = (a \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b}) \wedge (a \vee b)$

$$\begin{array}{ll} \overline{a} \rightarrow \overline{b} & b \rightarrow a \\ a \rightarrow b & \overline{b} \rightarrow \overline{a} \\ a \rightarrow \overline{b} & b \rightarrow \overline{a} \\ \overline{a} \rightarrow b & \overline{b} \rightarrow a \end{array}$$

Recall that implications are transitive: If $x \rightarrow y$ and $y \rightarrow z$, then $x \rightarrow z$

Satisfiability of implications

If there is a chain of implications $x \to a \to \cdots \to \overline{x}$, then $x = F$
If there is a chain of implications $\overline{x} \to b \to \cdots \to x$, then $x = T$

If both chains of implications exist, then the set of implications is not satisfiable (because a literal cannot be both true and false)

Thus, if we start with a formula in 2-CNF and write out the set of equivalent implications and find $x \to \overline{x}$ and $\overline{x} \to x$ for some variable x, then the formula is not satisfiable

Satisfiability of implications

If there is a chain of implications $x \to a \to \cdots \to \overline{x}$, then $x = F$
If there is a chain of implications $\overline{x} \to b \to \cdots \to x$, then $x = T$

If both chains of implications exist, then the set of implications is not satisfiable (because a literal cannot be both true and false)

Thus, if we start with a formula in 2-CNF and write out the set of equivalent implications and find $x \to \overline{x}$ and $\overline{x} \to x$ for some variable x, then the formula is not satisfiable

In fact, this condition is necessary, not merely sufficient for a formula to be unsatisfiable (harder to prove (Krom 1967))

That is, a formula is unsatisfiable iff $x \to \overline{x}$ and $\overline{x} \to x$ for some variable x

Turning a formula into a directed graph

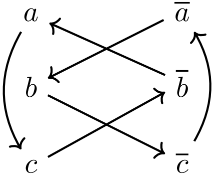
If the formula has m clauses and n variables, then we can construct the formula's implication graph which has $2n$ vertices and $2m$ edges

Let the vertices of the graph be each variable and its negation (i.e., x and $\overline{x}$ are vertices for each variable x)

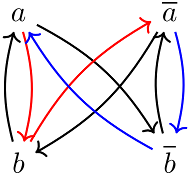
Let (x, y) be a directed edge in the graph for each implication $x \to y$

There's a path from x to y in the graph iff there is a chain of implications $x \to a \to \cdots \to y$

$\phi = (a \vee b) \wedge (\overline{a} \vee c) \wedge (\overline{b} \vee \overline{c})$:



$\psi = (a \vee \overline{b}) \wedge (\overline{a} \vee b) \wedge (\overline{a} \vee \overline{b}) \wedge (a \vee b)$:



$a \to b \to \overline{a}$ $\quad$ $\overline{a} \to \overline{b} \to a$

2-SAT $\in$ P

Now we can use our polynomial-time decider for PATH to decide $2\text{-}\mathrm{SAT}$ in polynomial time

Let R decide PATH and construct D to decide $2\text{-}\mathrm{SAT}$
D = "On input $\langle \phi \rangle$,

1. Construct the implication graph G for ϕ
2. For each variable x in ϕ,
3. Run R on $\langle G, x, \overline{x} \rangle$ and $\langle G, \overline{x}, x \rangle$; if R accepts both, then *reject*
4. Otherwise *accept*"

2-SAT $\in$ P

Now we can use our polynomial-time decider for PATH to decide 2-SAT in polynomial time

Let R decide PATH and construct D to decide 2-SAT

D = "On input $\langle \phi \rangle$,

1. Construct the implication graph G for ϕ
2. For each variable x in ϕ,
3. Run R on $\langle G, x, \overline{x} \rangle$ and $\langle G, \overline{x}, x \rangle$; if R accepts both, then *reject*
4. Otherwise *accept*"

$\langle \phi \rangle \notin$ 2-SAT iff ϕ is unsatisfiable iff there is some variable x such that there is a path from x to $\overline{x}$ and a path from $\overline{x}$ to x in the implication graph iff D rejects

Since PATH $\in$ P, R runs in time polynomial in its input $\langle G, s, t \rangle$ which has size polynomial in the size of $\langle \phi \rangle$

Constructing G takes polynomial time in the size of $\langle \phi \rangle$ and R is run a polynomial number of times (twice per variable) so D runs in polynomial time. Therefore, 2-SAT $\in$ P

Why is constructing the graph polynomial time?

Remember, if ϕ has m clauses and n variables, then G has $2n$ vertices and $2m$ edges

For example, we could use the adjacency matrix representation which would be a $2n \times 2n$ matrix

Recap

$\mathrm{PATH} \in \mathrm{P}$ because we were able to give a polynomial time decider for it

By naïvely enumerating all 2^n possible truth values, we showed $2\text{-}\mathrm{SAT} \in \mathrm{EXPTIME}$

By being more clever and constructing a graph corresponding to formulae in 2-CNF, we showed $2\text{-}\mathrm{SAT} \in \mathrm{P}$

Can we always be more clever?

Sadly, no. $\mathrm{P} \subsetneq \mathrm{EXPTIME}$

That is, there are problems (equivalently languages) that require exponential time to decide

Here's one: $A = \{\langle M, w, 1^k \rangle \mid M$ is a TM that accepts w in at most 2^k steps$\}$

$A \in \mathrm{EXPTIME}$: Simulate running M on w for 2^k steps takes exponential time

$A \notin \mathrm{P}$: Harder to prove, but true