**PS10 due next Friday, Apr 25.**
**PRR12 coming later today.**

# Class 24:
## Poly-time reductions
## Class NP

University of Virginia
cs3120: DMT2
Wei-Kai Lin

# Recap: TIME Complexity Classes

$TIME_{\mathsf{TM}}\big(T(n)\big)$ is the set of Boolean functions for which a Turing Machine $M$ exists such that $M$ halts after at most $T(n)$ steps for all $n$-bit input and $M$ computes the function.

# Recap: Complexity Class: P

A class for "Polynomial Time"

Class **P** = $\bigcup_{c\in\mathbb{N}} TIME_{TM}(n^c)$

**P**$_{TM}$ = $\bigcup_{c\in\mathbb{N}} TIME_{TM}(n^c)$ — One-way infinite tape TM

**P**$_{TM2w}$ = $\bigcup_{c\in\mathbb{N}} TIME_{TM2w}(n^c)$ — Two-way infinite tape TM

**P**$_{\mathrm{RAM}}$ = $\bigcup_{c\in\mathbb{N}} TIME_{\mathrm{RAM}}(n^c)$ — **RAM** machine

**P**$_{\mathrm{Python}}$ = $\bigcup_{c\in\mathbb{N}} TIME_{\mathrm{Python}}(n^c)$ — Idealized **Python** interpreter

**P**$_{TM}$ = **P**$_{TM2w}$ = **P**$_{\mathrm{RAM}}$ = **P**$_{\mathrm{Python}}$

$TIME_{\mathrm{TM}}(T(n))$ is the set of Boolean functions for which a Turing Machine $M$ exists such that $M$ halts after at most $T(n)$ steps for all $n$-bit input and $M$ computes the function.

# Worst-case vs. Average case

$TIME_{TM}(T)$ and $P, EXP$ are defined based on **worst-case** running time on all inputs.

Suppose an algorithm $A$ for $F$ solves all instance $x \in \{0,1\}^n$ in linear $\Theta(n)$ except one instance, for which it takes $2^n$ to solve. Then, what is the **average** (case) running time for a random $x \in \{0,1\}^n$ ?

(This is different from expected running time for quick sort.)

# Problem: LongestPath

# Functions in P?

***LongestPath***

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$, and a path length, $\ell \in \mathbb{N}$.
**Output:** If there is a simple path from $s$ to $t$ in $G$ of length at least $\ell$, $1$. Otherwise, $0$.

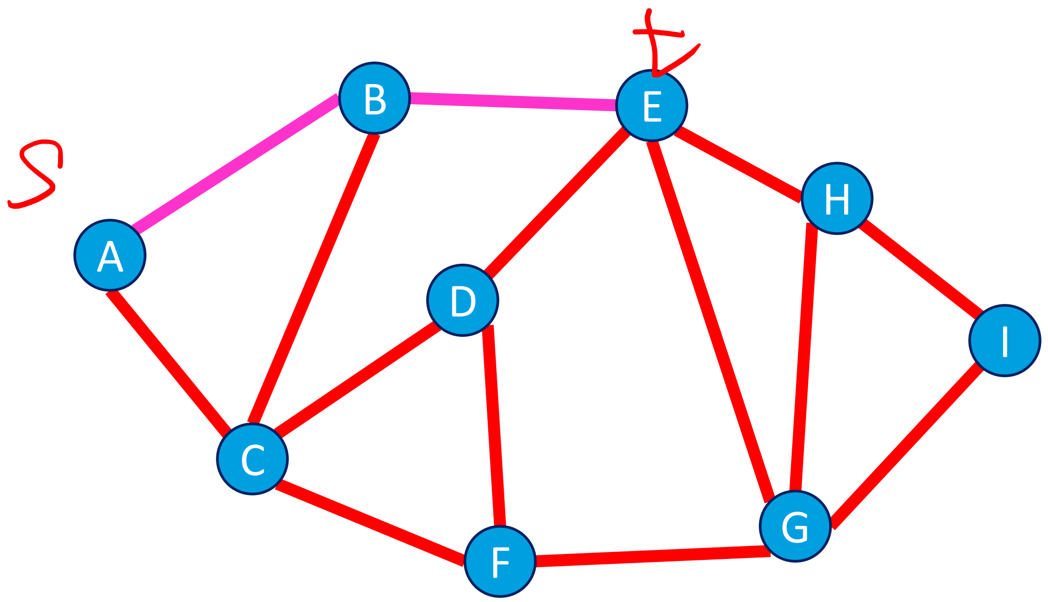
LongestPath is **not known** to be in **P**: it might be, it might not be. (We'll see a more about this in future classes…)

Definition: a *simple path* in a graph $G = (V, E)$ from $s, t \in V$ is a path from $s$ to $t$ where no node is repeated.

Definition: a ***path*** from $v_1$ to $v_k$ in a graph $G = (V, E)$ is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ such that $\forall\ 1 \leq i \leq k-1$, $(v_i, v_{i+1}) \in E$. A ***simple path*** is a path with no repeated nodes.

# Shortest Path

Given an unweighted graph, start node $s$ and an end node $t$, how long is *shortest path* from $s$ to $t$?

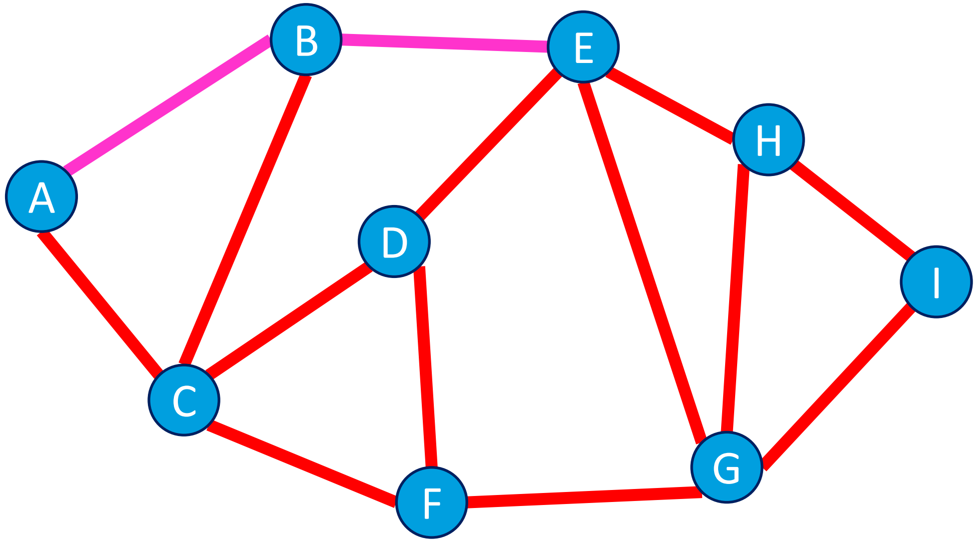



ShortestPath-Search($G$, **A**, **E**) = 2

Is *ShortestPath* in **P**?

# Shortest Path

ShortestPath$(G, s, t, k)$:
**1** iff there is a path from s to t in G with $\leq k$ steps

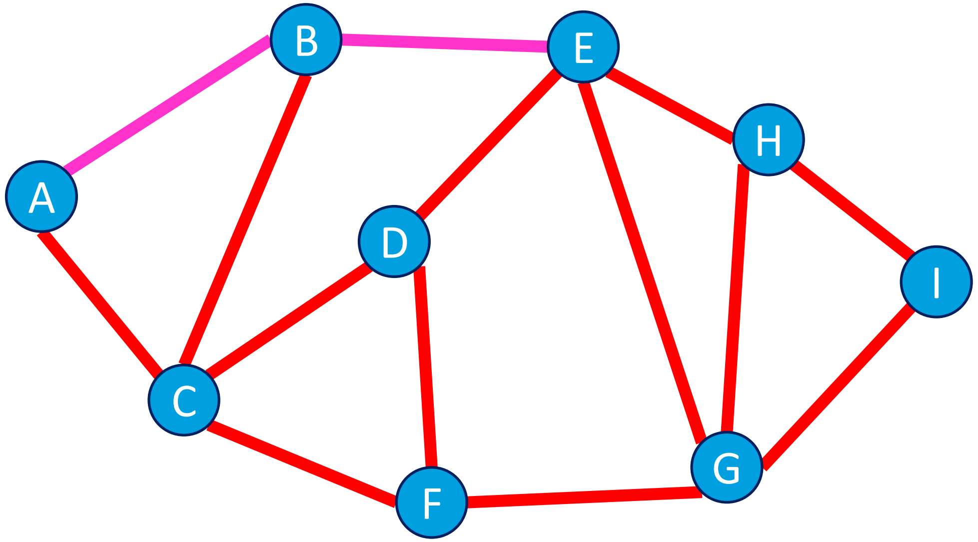



ShortestPath-Search($G$, **A**, **E**) = 2

Is *ShortestPath* in **P**?

# Breadth First Search

ShortestPath$(G, s, t, k)$:
**1** iff there is a path from s to t in G with $\leq k$ steps




Python Running time: $O(|V| + |E|)$

TM poly$(|V|+|E|)$

ShortestPath $\in$ **P**

Definition: a ***path*** from $v_1$ to $v_k$ in a graph $G = (V, E)$ is a sequence of nodes $(v_1, v_2, \dots, v_k)$ such that $\forall\ 1 \leq i \leq k-1$, $(v_i, v_{i+1}) \in E$. A ***simple path*** is a path with no repeated nodes.

# Longest Path

Given a start node $s$ and an end node $t$, how long is longest *simple* path from $s$ to $t$?

*LongestPath-Search*$(G, s, t)$

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$

**Output:** The length of the longest simple path from $s$ to $t$ in $G$.

# Longest Path Algorithm

$n = |V|$

maxlength = 0

for all subsets S of V: $2^{|V|}$ $\quad 2^{|V|} \cdot |V|!$

    for all orderings of S: $|S|! \leq |V|!$ $\quad 2^n \cdot 2^{n \log n}$

        if it is a path from $s$ to $v$: $\quad \leq 2^{n + n \log n}$

            if len(S) > maxlength: maxlength = len(S)

return maxlength

**What's the running time?**

# Is *LongestPath* $\in$ EXP?

*LongestPath-Search*

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$

**Output:** The length of the longest simple path from $s$ to $t$ in $G$.

# Is *LongestPath* ∈ EXP?

*LongestPath-Search*

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$

**Output:** The length of the longest simple path from $s$ to $t$ in $G$.

**Definition 13.2 ($\mathbf{P}$ and $\mathbf{EXP}$)**

Let $F : \{0,1\}^* \to \{0,1\}$. We say that $F \in \mathbf{P}$ if there is a polynomial $p : \mathbb{N} \to \mathbb{R}$ and a Turing machine $M$ such that for every $x \in \{0,1\}^*$, when given input $x$, the Turing machine halts within at most $p(|x|)$ steps and outputs $F(x)$.

We say that $F \in \mathbf{EXP}$ if there is a polynomial $p : \mathbb{N} \to \mathbb{R}$ and a Turing machine $M$ such that for every $x \in \{0,1\}^*$, when given input $x$, $M$ halts within at most $2^{p(|x|)}$ steps and outputs $F(x)$.

$2^{p(n)}$

$p(n) = n$

$n^2$

$n \log n$

$n^4$

# How can we turn *LongestPath* into a *decision* problem?

*LongestPath-Search*

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$

**Output:** The $\underbrace{\text{length of}}_{\text{int}}$ the longest simple path from $s$ to $t$ in $G$.

$$F : \{0,1\}^* \to \underbrace{\{0,1\}}_{\text{bit}}$$

# How can we turn *LongestPath* into a *decision* problem?

*LongestPath-Search*
**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$
**Output:** The length of the longest simple path from $s$ to $t$ in $G$.

$F: \{0,1\}^* \to \{0,1\}$

*LongestPath-Decision*$(G, s, t, \ell)$

**Input:**

A finite graph $G = (V, E)$, two vertices, $s, t \in V$, and a path length, $\ell \in \mathbb{N}$.

**Output:**

If there is a simple path from $s$ to $t$ in $G$ of length at least $\ell$, 1. Otherwise, 0.

# Is *LongestPath-Decision* easier than *LongestPath-Search*?

*LongestPath-Decision*$(G, s, t, \ell)$:

    if *LongestPath-Search* $(G, s, t) \geq \ell$:

        return 1

    return 0

*LongestPath-Decision* is easier or equal to *LongestPath-Search*

# Can we use *LongestPath-Decision* to solve *LongestPath-Search*?

*LongestPath-Search* $(G, s, t)$

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$

**Output:** The length of the longest simple path from $s$ to $t$ in $G$.

$\in [0, n = |V|]$

*LongestPath-Decision* $(G, s, t, \ell)$

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$, and a path length, $\ell \in \mathbb{N}$.

**Output:** If there is a simple path from $s$ to $t$ in $G$ of length at least $\ell$, 1. Otherwise, 0.

binary search $(0, n)$

for $\ell = 0 \ldots n$

if LP-Decision$(\cdots \ell) = 0$

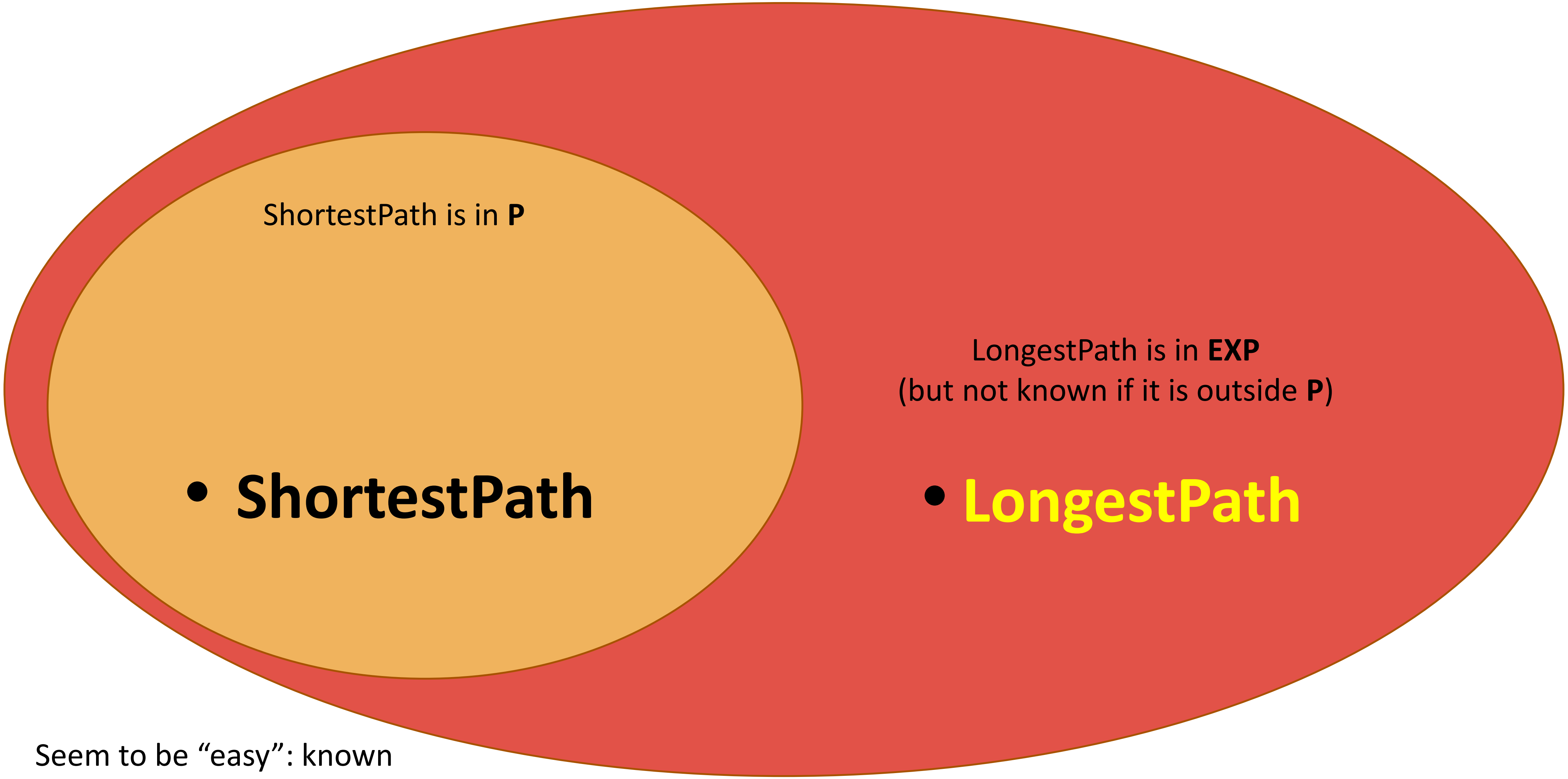
return $\ell - 1$

# *Search* and *Decisional LongestPath* are "equivalent in poly-time"

?  ?

*LongestPath-Search* ∈ **P** iff *LongestPath-Decision* ∈ **P**

*LongestPath-Search* ∈ **EXP** and *LongestPath-Decision* ∈ **EXP**




Seem to be "easy": known polynomial-time algorithms

Class **P** = $\bigcup_{c \in \{1,2,3,\dots\}} TIME_{TM}(n^c)$

Class **EXP** = $\bigcup_{c \in \{1,2,3,\dots\}} TIME_{TM}(2^{n^c})$

# Polynomial-Time Reductions

# Recap: Computability Reductions

$$A \leq_R B$$

A reduces to B

We can prove a problem **B** is uncomputable by showing that if we could compute **B** we could use the machine that computes **B** to compute **A**, which we already know is uncomputable.

Since we were showing ***uncomputability***, the reduction can do anything that can be **computed by a TM**.

# Polynomial-Time Reductions

$$A \leq_P B$$

A (polynomial-time) *reduces* to B

We will use this to prove that problem **B** is as least as "hard" as problem **A** by showing that if computing **B** is in **P** we could use the machine that computes **B** to compute **A** in **P** (but we already know **A** is "hard").

We haven't yet defined "hard" here, or shown any problem we know is "hard" (preview: *LongestPath* is "hard").

Since we are showing a hardness property related to **P**, the reduction must only involve computation that can be done in **P**.

# Comparison

$$A \leq_R B$$

A reduces to B

$$A \leq_P B$$

A (polynomial-time) *reduces* to B

Prove non-computability of **B**

Reduction must be computable

Prove non-"easiness" of **B**

Reduction must be "easy" (in **P**)

# Polynomial Reduction Definition

**Definition 14.1 (Polynomial-time reductions)**

Let $F, G : \{0,1\}^* \rightarrow \{0,1\}$. We say that $F$ *reduces to* $G$, denoted by $F \leq_p G$ if there is a polynomial-time computable $R : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every $x \in \{0,1\}^*$,

$$F(x) = G(R(x)) \; . \quad (14.1)$$

We say that $F$ and $G$ have *equivalent complexity* if $F \leq_p G$ and $G \leq_p F$.

Note: This is the "Karp reduction" definition. (There are restrictive definitions – see PS10 Problem 2)

# Trivial Example

$F, G: \{0,1\}^* \to \{0,1\}$. $F \le_p G$ if there is a polynomial-time computable $R: \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*, F(x) = G(R(x))$.

Prove

$$\overset{F}{LongestPathDecision} \le_P \overset{G}{LongestPathDecision}.$$

$R(x) = x$

$F(x) = \cancel{G(G(x} G(R(x)) = G(x) = F(x)$

How about proving:

$LongestPathDecision \le_P LongestPathSearch$?

# Less Trivial Example

$F, G: \{0,1\}^* \to \{0,1\}$. $F \leq_p G$ if there is a polynomial-time computable $R: \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $F(x) = G(R(x))$.

Prove $ODD \leq_P EVEN$.

F G

$ODD$: $\{0,1\}^* \to \{0,1\}$. Output 1 of input is binary representation of a odd natural number, 0 otherwise.
$EVEN$: $\{0,1\}^* \to \{0,1\}$. Output 1 of input is binary representation of a even natural number, 0 otherwise.

$R(x) = x+1$

$ODD(x) = Even(R(x))$
$= E(x+1)$

$NOT(EVEN(x))$

?

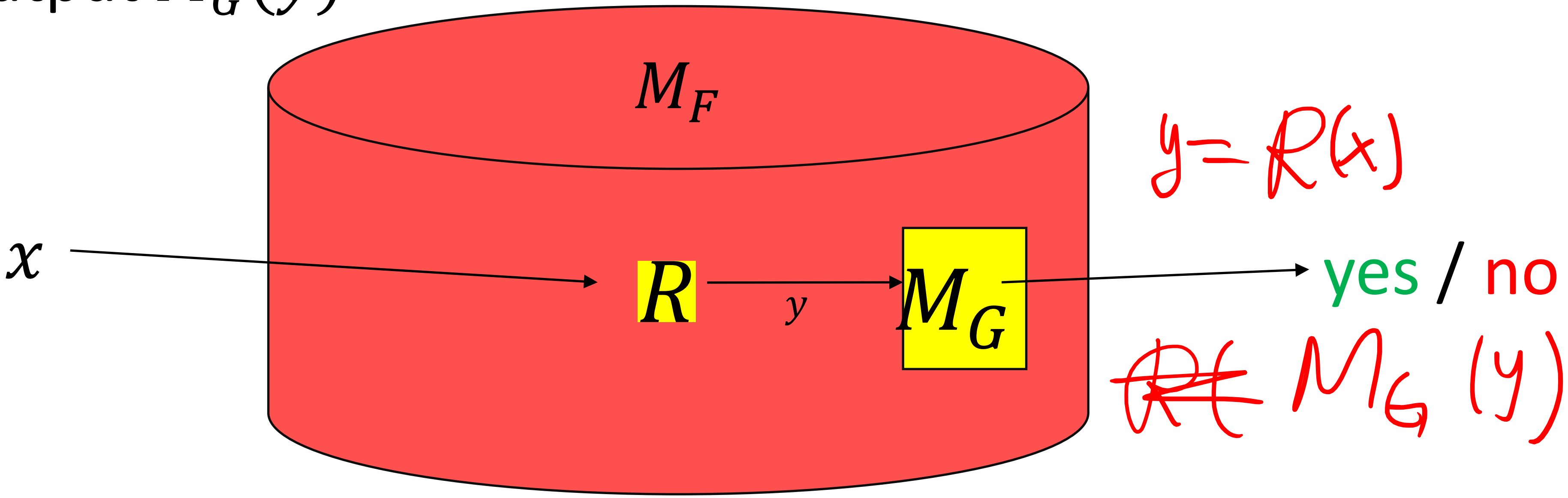
# More poly-time reductions

# The picture for Karp reductions

We have a solver $M_G$ for $G$

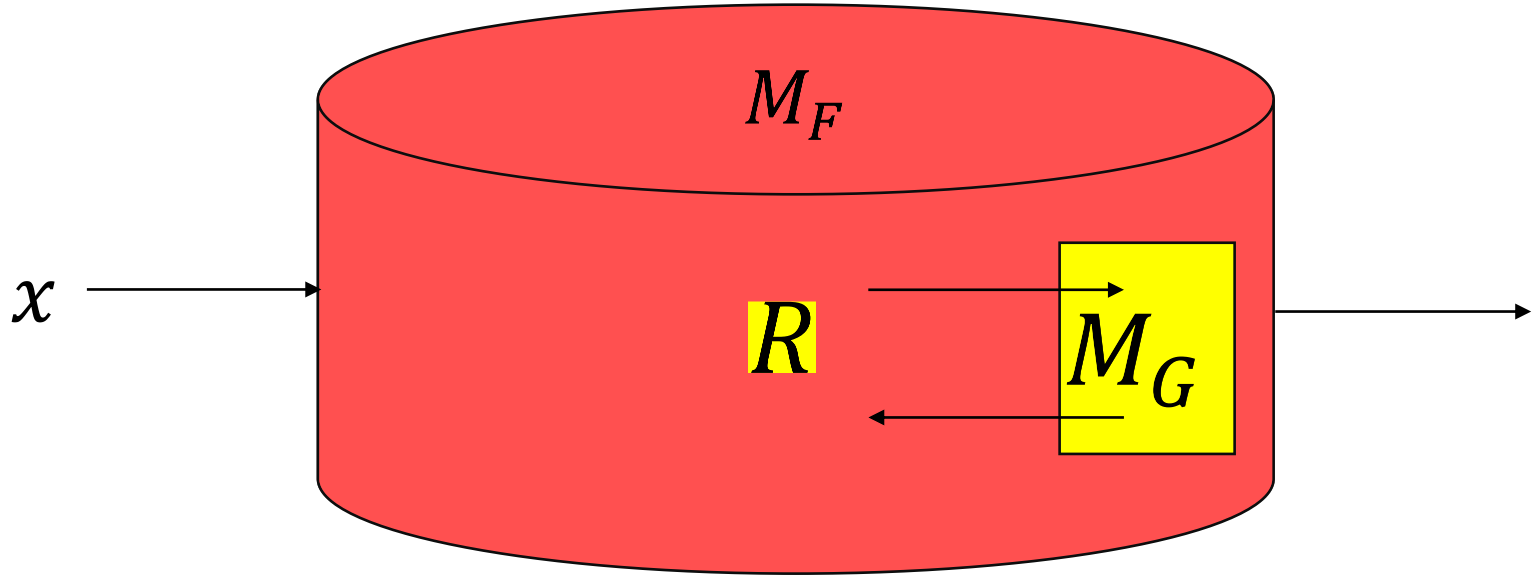
We design a solver $M_F$ for $F$ as follows:

1. Use the poly-time reduction $R$ to modify input $x$ into $y$
2. Output $M_G(y)$



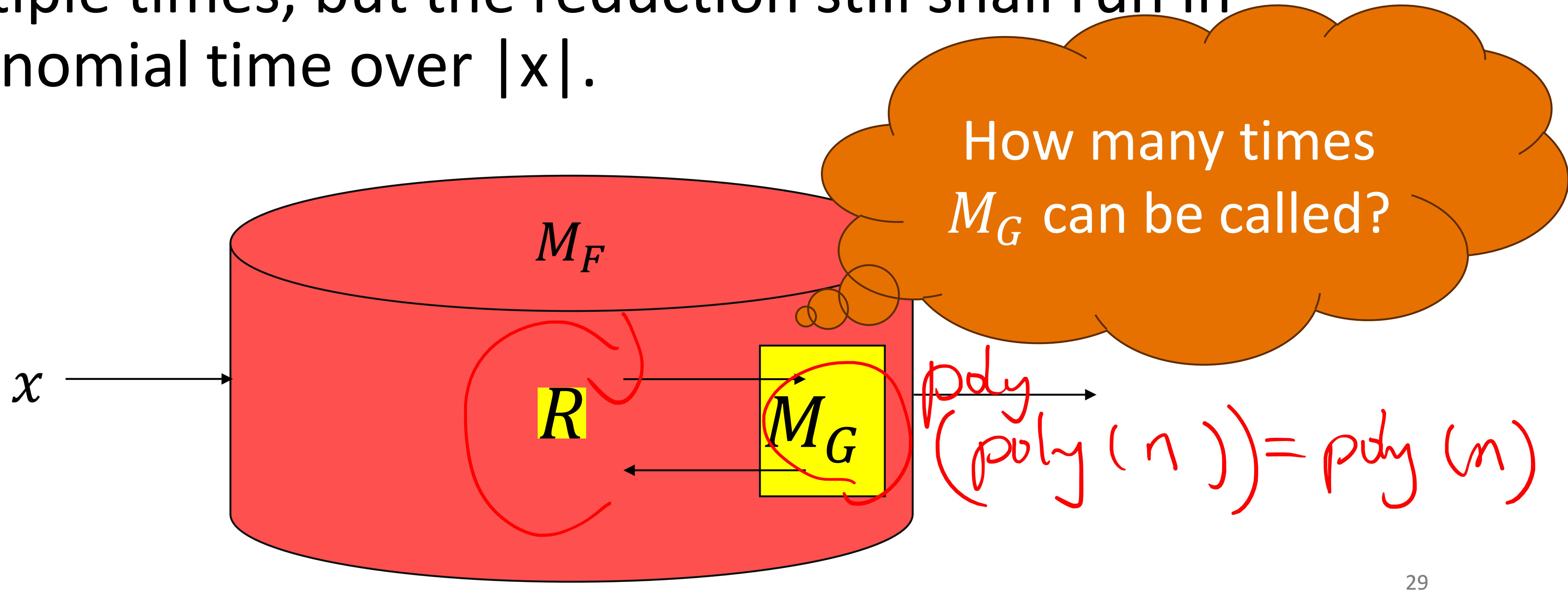
# Cook reductions

In Cook reductions, the “subroutine” $M_G$ can be used multiple times, but the reduction still shall run in polynomial time over |x|.



# Cook reductions

In Cook reductions, the “subroutine” $M_G$ can be used multiple times, but the reduction still shall run in polynomial time over |x|.



# Examples

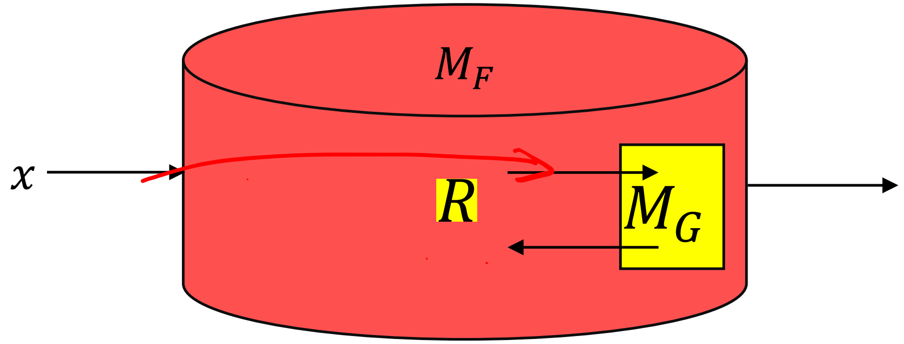
$F$ $G$

$LongestPathDecision \leq_P LongestPathSearch.$

$R: (\ell)$

$maxL = M_G(\ldots)$

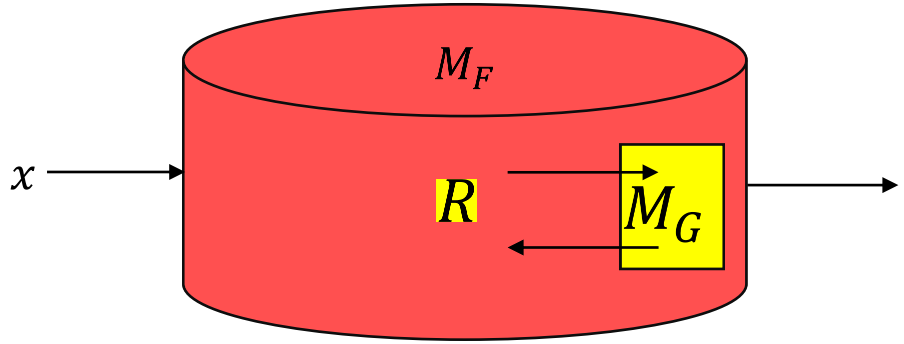
ret if $maxL \geq \ell$




$LongestPathSearch \leq_P LongestPathDecision.$

for $\ell = 0 \ldots n$

$M_G(\ldots)$



# Karp Reductions vs. Cook Reductions

Every Karp reduction is a Cook reduction too.

A Cook reduction is not necessarily a Karp reduction.

# Uses of Reductions

Reductions can be used to show easiness and hardness

Suppose $F \leq_P G$ then we have:

- $G \in \mathbf{P} \rightarrow F \in \mathbf{P}$ (deriving easiness)
- $F \notin \mathbf{P} \rightarrow G \notin \mathbf{P}$ (deriving hardness)
  Michael Sipser: "whistling pigs → flying horses"

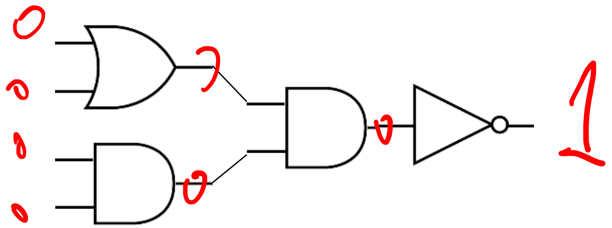
# More “Hard” Problems: SAT

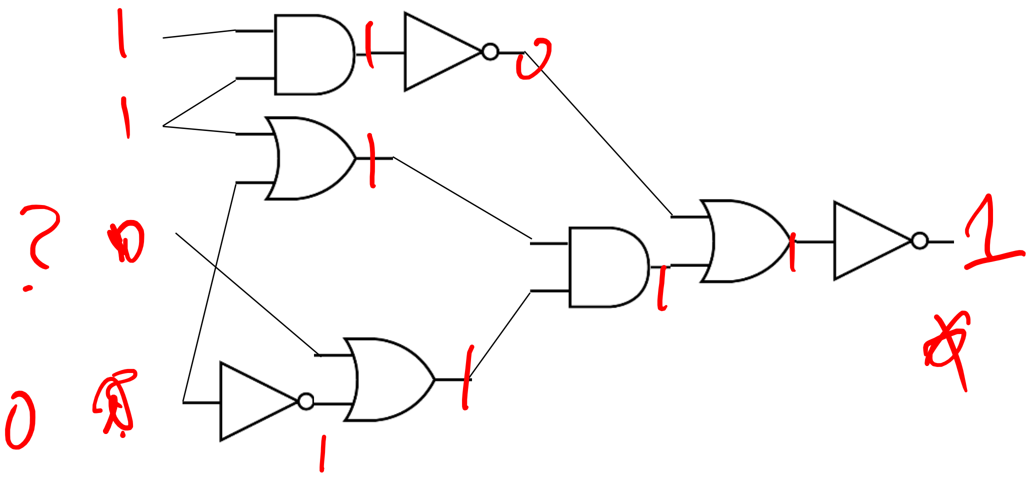
# Another Problem (Seemingly) Outside P

CircuitSAT :
$\{C : C$ is a Boolean circuit such that $\exists x,\ C(x) = 1\}$

# Another Problem (Seemingly) Outside P

CircuitSAT :
$\{C : C \text{ is a Boolean circuit such that } \exists x,\ C(x) = 1\}$





# 3-CNF (Conjunctive Normal Form)

3-CNF formula is logical AND of clauses, each an OR of 3 literals.

not$(x_2)$

$$F(x_1, x_2, x_3, x_4) =$$

$$\underbrace{(x_1 \vee x_2 \vee x_3)}_{\textbf{Clause}} \wedge (x_1 \vee \overline{x_2} \vee x_2) \wedge (x_4 \vee x_2 \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_1} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

Variables

$x_i$ $\quad i \in [n]$

# 3-CNF

Def: Suppose $x_1, \dots x_n$ are Boolean **variables**.

Any $x_i$ or $\overline{x_i} = 1 - x_i$ is a **literal**.

Any $(z_i \lor z_j \lor z_k)$ over literals $z_i, z_j, z_k$ is a **3-clause**.

A **3-CNF** (conjunctive normal form) formula looks like $C_1 \land C_2 \land \cdots C_m$ in which all $C_i$ are 3-clauses.

We say assignment $x$ to $x_1, \dots x_n$ satisfies 3-CNF $F$, denoted by $F(x) = 1$, if at least one literal in each clause is TRUE.

# 3-SAT (Satisfiability)

Given a 3-CNF formula $F$, is there an assignment of true/false such that $F$(assignment)=1?

$F(x_1, x_2, x_3, x_4) =$
$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_2) \wedge (x_4 \vee x_2 \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_1} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

$$\boldsymbol{x_1 = true}$$
$$\boldsymbol{x_2 = false}$$
$$\boldsymbol{x_3 = false}$$
$$\boldsymbol{x_4 = true}$$

Definition:

3-SAT(F) = 1 if there exists x such that F(x) = 1
0 otherwise.

# Examples of 3-CNF

Are they in 3-SAT?

- $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_3 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3)$

- $(x_1 \vee x_1 \vee x_1) \wedge (\bar{x}_1 \vee \bar{x}_1 \vee \bar{x}_1)$ $\notin$ 3-SAT

  $x_1$ $\bar{x}_1$

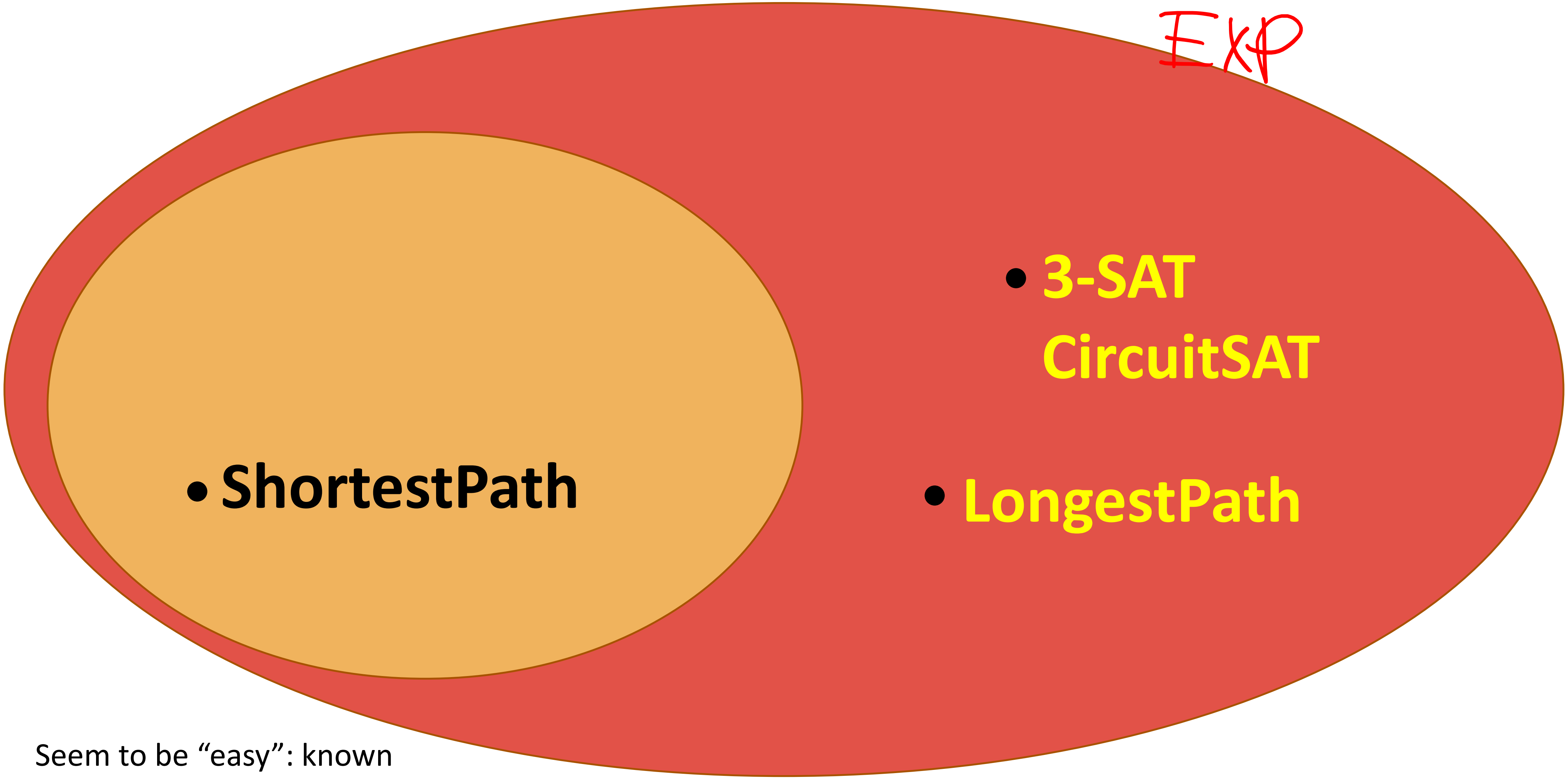
- $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$

# 3-SAT $\leq_P$ CircuitSAT

Suppose we are given a 3-CNF formula $F$. How do we (efficiently) transform $F$ into a circuit $C_F$ such that $C_F$ is satisfiable if and only if $F$ is satisfiable?

R( F )

return ~~S~~ CircuitSAT ( F )




Seem to be "easy": known polynomial-time algorithms

Class **P** = $\bigcup_{c \in \{1,2,3,\dots\}} TIME_{TM}(n^c)$

Class **EXP** = $\bigcup_{c \in \{1,2,3,\dots\}} TIME_{TM}(2^{n^c})$

# Complexity Class NP

NP stands for "non-deterministic" polynomial time.

# Revisiting the hard problems we have seen

What do these problems have in common?
CircuitSAT, LongestPath, 3-SAT

- We do not know any poly-time algorithm for them.
- In class **EXP**
- There are some poly-time reductions between them

Big idea: Once the instance is in the language,
there is a short convincing “witness” / “proof” for this claim.

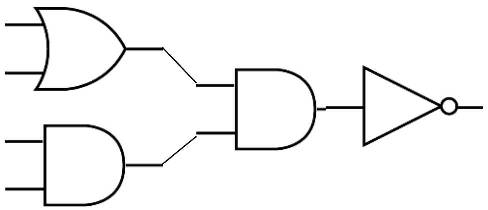
$F(x) = 1$

3-SAT: $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_3 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3)$

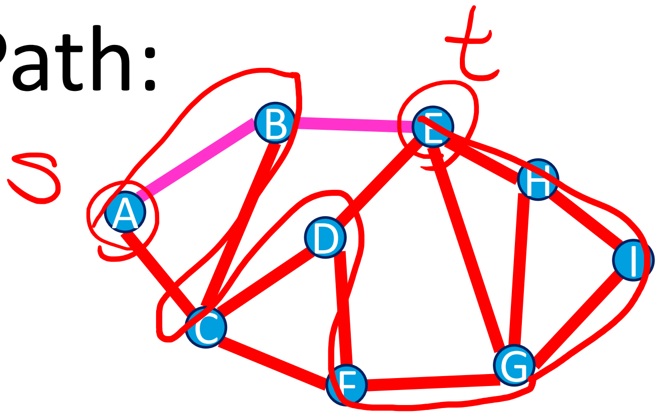
$w = 1 \quad 1 \quad 0 \quad 1 \quad 1$

CircuitSAT:

$w = 0\,0\,0\,0$



LongestPath:




$w = A\ BC\ DFG \cdots E$

# Defining Complexity Class NP

Boolean Func

Informal def: decisional problem $F$ is in $NP$ if: whenever a problem instance $x \in F$ then this can be proved by providing a **witness** that is **polynomial-time verifiable**.

# Defining Complexity Class NP

Formal Definition of NP:

problem SP, 3SAT, ...

**Definition 15.1 (NP)**

We say that $F : \{0,1\}^* \to \{0,1\}$ is in $\mathbf{NP}$ if there exists some integer $a > 0$ and $V : \{0,1\}^* \to \{0,1\}$ such that $V \in \mathbf{P}$ and for every $x \in \{0,1\}^n$,

$$F(x) = 1 \Leftrightarrow \underline{\exists_{w \in \{0,1\}^{n^a}}} \text{ s.t. } V(xw) = 1 \,. \quad (15.1)$$

$\forall x \quad \exists w(x)$

# The Class P

$F \leq_p G$ & $F \in NP$

Functions that can be computed in polynomial time by a standard Turing Machine.

& $G \in NP$

$\Rightarrow F \in NP$

$$\bigcup_{c \in \mathbb{N}} TIME_{TM}(n^c)$$

# The Class NP

Functions that can be **verified** in polynomial time by a standard Turing Machine.

Correctness of a **1** output can be *verified* in polynomial time given a witness.

A function $F: \{0,1\}^* \rightarrow \{0,1\}$ is in **NP** if there exists some $a \in \mathbb{N}^+$ and $V: \{0,1\}^* \rightarrow \{0,1\}$ such that $V \in \mathbf{P}$ and $\forall x \in \{0,1\}^n, F(x) = 1 \leftrightarrow \exists w \in \{0,1\}^{n^a}$ such that $V(x,w) = 1$.

Graph Isom $\in$ NP

GraphNonIsom $\overset{?}{\in}$ NP

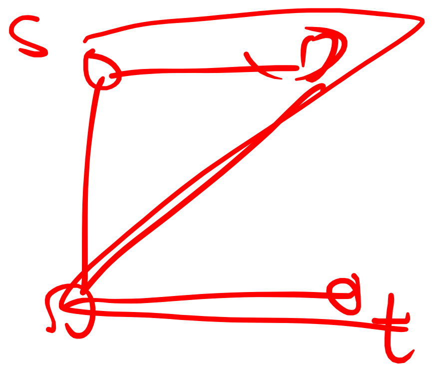
# *LongestPath* ∈ **NP**

***LongestPath***

**Input:** A finite graph $G = (V, E)$, two vertices, $s, t \in V$, and a path length, $n \in \mathbb{N}$.
**Output:** If there is a simple path from $s$ to $t$ in $G$ of length at least $n$, 1. Otherwise, 0.

A function $F: \{0,1\}^* \to \{0,1\}$ is in **NP** if there exists some $a \in \mathbb{N}^+$ and $V: \{0,1\}^* \to \{0,1\}$ such that $V \in \mathbf{P}$ and $\forall x \in \{0,1\}^n, F(x) = 1 \leftrightarrow \exists w \in \{0,1\}^{n^a}$ such that $V(x, w) = 1$.

Correctness of a **1**-output can be *verified* in polynomial time given a witness.




$l = 4$

$l = 5$ No Witness

# 3SAT ∈ **NP**

**3SAT**

**Input:** A Boolean formula in 3CNF form.

**Output:** If there is an assignment of values to variables that makes the formula to True, 1. Otherwise, 0.

A function $F: \{0,1\}^* \to \{0,1\}$ is in **NP** if there exists some $a \in \mathbb{N}^+$ and $V: \{0,1\}^* \to \{0,1\}$ such that $V \in \mathbf{P}$ and $\forall x \in \{0,1\}^n, F(x) = 1 \leftrightarrow \exists w \in \{0,1\}^{n^a}$ such that $V(x,w) = 1$.

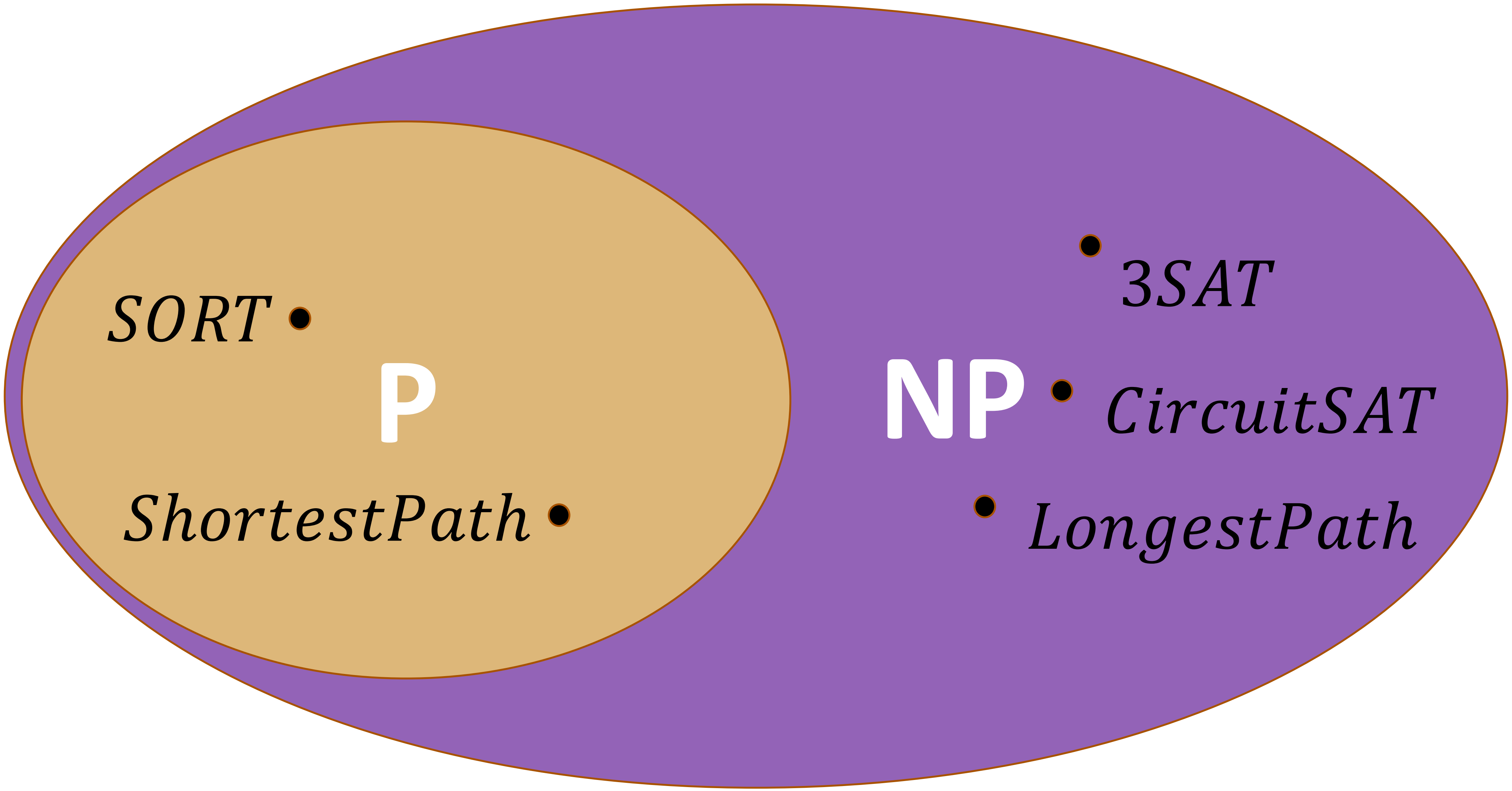
Correctness of a 1-output can be *verified* in polynomial time given a witness.

$g(x) = (x_1 \vee x_2 \vee x_4) \wedge ( \ldots ) \wedge ( - )$

$x_1 = 1$

$x_2 = 1$

$\vdots$




Unknown if $3SAT \in$ **P**

Known that $3SAT \in$ **NP**

# Charge

**Time complexity**

*Polynomial-time reductions*

*Class* **NP**

**PS10 due next Friday, Apr 25**