

CMSC 28100

# Introduction to Complexity Theory

Autumn 2025

Instructor: William Hoza



# Homework reminder

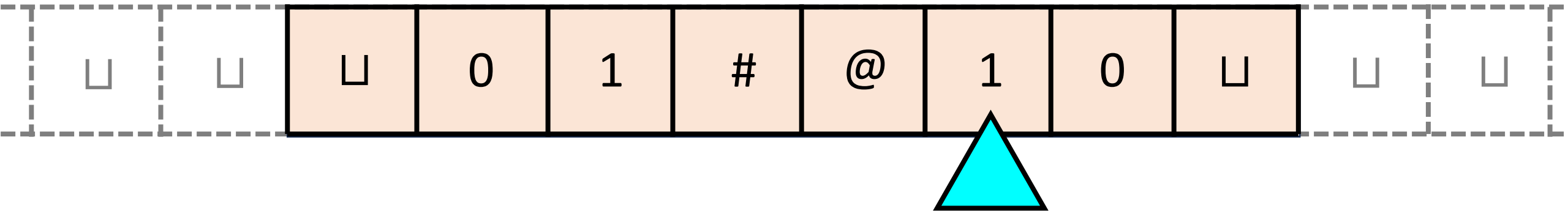
- Exercises 1-3 are due **this Friday (October 3) at 11:59pm**
- If you joined the course late and you need an extension, send me an email

# Office hours / student meet-up time

- Thursdays 11am to noon: TA office hours (Mirza)
- Thursdays 2pm to 3pm: Student meet-up time
- Thursdays 3pm to 4pm: TA office hours (Zelin)
- Fridays 9am to 11am: My office hours

Which problems  
can be solved  
through computation?

# The Turing machine model




# Defining Turing machines rigorously

- **Definition:** A **Turing machine** is a 7-tuple  $M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \sqcup, \delta)$

such that

- $Q$  is a finite set (the set of “states”)
- $q_0, q_{\text{accept}}, q_{\text{reject}} \in Q$  and  $q_{\text{accept}} \neq q_{\text{reject}}$
- $\Sigma$  is a finite set of symbols (the “tape alphabet”)
- $\sqcup$  is a symbol (the “blank symbol”)
- $\{0, 1, \sqcup\} \subseteq \Sigma$  and  $\sqcup \notin \{0, 1\}$
- $\delta$  is a function  $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$  (the “transition function”)

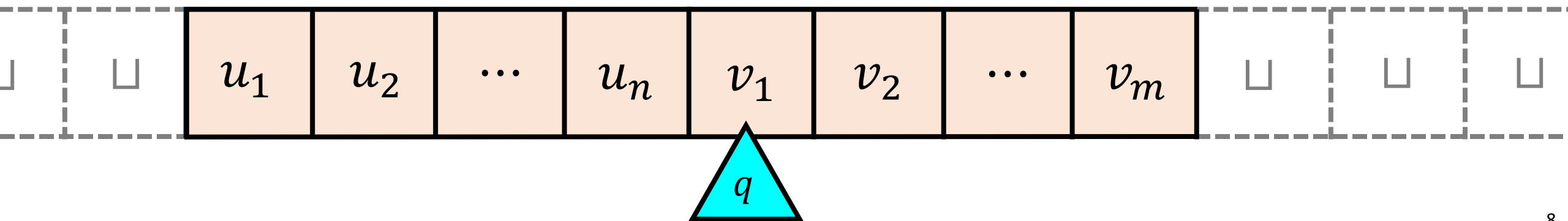
 Warning: The definition in the textbook is slightly different. Sorry!  
(The two models are equivalent.)

# Defining TM computation rigorously

- Transition function  $\delta$  describes the **local** evolution of the computation
- What about the **global** evolution?

# Configurations of a Turing machine

- Let  $M = (Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \sqcup, \delta)$  be a Turing machine
- A **configuration** of  $M$  is a triple  $(u, q, v)$  where  $u \in \Sigma^*$ ,  $q \in Q$ ,  $v \in \Sigma^*$ , and  $v \neq \epsilon$ . Interpretation:
  - The tape currently contains  $uv$
  - The machine is currently in state  $q$  and the head is pointing at the first symbol of  $v$



# Configuration shorthand

- Instead of  $(u, q, v)$ , we often write  $uqv$
- Think of  $uqv$  as a string over the alphabet  $\Sigma \cup Q$
- Only works if  $Q \cap \Sigma = \emptyset$ 
  - Can assume without loss of generality (rename states if necessary)

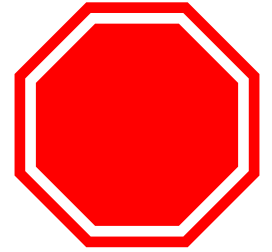
# The initial configuration

- Let  $w \in \{0, 1\}^*$  be an input
- The initial configuration of  $M$  on  $w$  is  $q_0 \sqcup w$

# The “next” configuration

- For any configuration  $uqv$ , we define  $\text{NEXT}(uqv)$  as follows:
  - Break  $uqv$  into individual symbols:  $uqv = u_1u_2 \dots u_{n-1}u_nqv_1v_2v_3 \dots v_m$
  - If  $\delta(q, v_1) = (q', b, \text{R})$ , then  $\text{NEXT}(uqv) = u_1u_2 \dots u_{n-1}u_nbq'v_2v_3 \dots v_m$ 
    - Edge case: If  $m = 1$ , then  $\text{NEXT}(uqv) = u_1u_2 \dots u_{n-1}u_nbq' \sqcup$
  - If  $\delta(q, v_1) = (q', b, \text{L})$ , then  $\text{NEXT}(uqv) = u_1u_2 \dots u_{n-1}q'u_nbv_2v_3 \dots v_m$ 
    - Edge case: If  $n = 0$ , then  $\text{NEXT}(uqv) = q' \sqcup bv_2v_3 \dots v_m$
- We write  $\text{NEXT}_M(uqv)$  if  $M$  is not clear from context

# Halting configurations

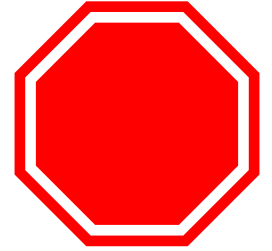


- An **accepting configuration** is a configuration of the form  $uq_{\text{accept}}v$
- A **rejecting configuration** is a configuration of the form  $uq_{\text{reject}}v$
- A **halting configuration** is an accepting or rejecting configuration

# Computation history

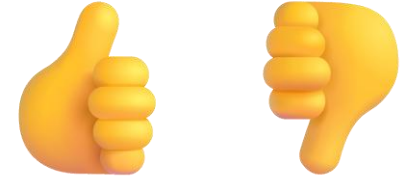
- Let  $w \in \{0, 1\}^*$  be an input
- Let  $C_0$  be the initial configuration of  $M$  on  $w$ , i.e.,  $C_0 = q_0 \sqcup w$
- Inductively, for each  $i \in \mathbb{N}$ , let  $C_{i+1} = \text{NEXT}(C_i)$
- The **computation history** of  $M$  on  $w$  is the sequence  $C_0, C_1, \dots, C_T$ , where  $C_T$  is the first **halting** configuration in the sequence
- If there is no such  $C_T$ , then the computation history is  $C_0, C_1, C_2, \dots$  (infinite)

# Halting and looping



- If the computation history of  $M$  on  $w$  is finite, we say  $M$  **halts** on  $w$
- Otherwise, we say  $M$  **loops** on  $w$

# Accepting and rejecting



- Suppose  $M$  halts on  $w$
- The computation history is finite,  $C_0, C_1, \dots, C_T$
- If  $C_T$  is an accepting configuration, we say  $M$  accepts  $w$
- If  $C_T$  is a rejecting configuration, we say  $M$  rejects  $w$

# Time



- Suppose the computation history of  $M$  on  $w$  is  $C_0, C_1, \dots, C_T$
- We say that  $T$  is the **running time** of  $M$  on  $w$
- If  $M$  loops on  $w$ , then its running time on  $w$  is  $\infty$
- We say that  **$M$  halts on  $w$  within  $T$  steps** if the running time of  $M$  on  $w$  is at most  $T$

# Space

- The **space used** by  $M$  on  $w$  is
  - (Can be  $\infty$ )

- Formally, let  $C_0, C_1, \dots$  be the (finite or infinite) computation history of  $M$  on  $w$
- Write  $C_i = (u_i, q_i, v_i)$  where  $u_i \in \Sigma^*$ ,  $q_i \in Q$ ,  $v_i \in \Sigma^*$
- The space used by  $M$  on  $w$  is  $\max_i |u_i v_i|$

Which of the following statements is false?

**A:** Space used on  $w$  is at most  $|w| + 1 + \text{running time on } w$

**B:** If  $M$  halts on  $w$  within  $|w|$  steps, then  $M$  halts on  $ww$

**C:** If  $M$  halts on  $w$ , then  $M$  uses a finite amount of space on  $w$

**D:** If  $M$  uses a finite amount of space on  $w$ , then  $M$  halts on  $w$

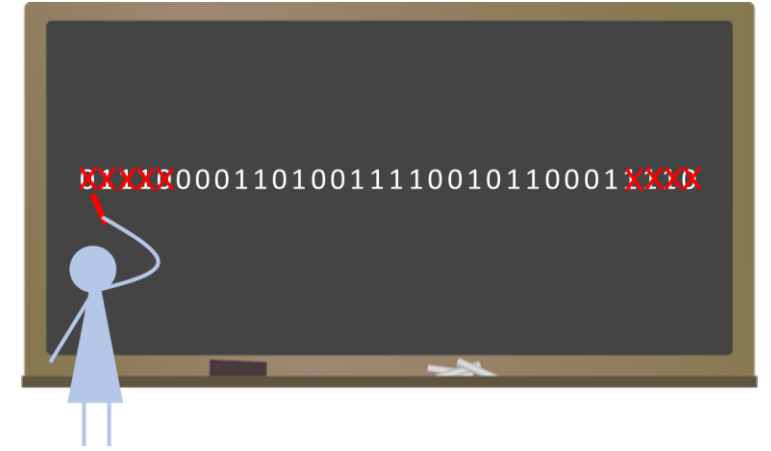
Respond at [PollEv.com/whoza](https://pollev.com/whoza) or text "whoza" to 22333

Which problems  
can be solved  
through computation?

# Deciding a language

- Let  $M$  be a Turing machine and let  $Y \subseteq \{0, 1\}^*$
- We say that  $M$  **decides**  $Y$  if
  - $M$  accepts every  $w \in Y$ , and
  - $M$  rejects every  $w \in \{0, 1\}^* \setminus Y$
- This is a mathematical model of what it means to “**solve a problem**”

# Example: Palindromes



- **Informal problem statement:** “Given  $w \in \{0, 1\}^*$ , determine whether  $w$  is the same forward and backward.”
- **The same problem, formulated as a language:**  
$$\text{PALINDROMES} = \{w \in \{0, 1\}^* : w \text{ is the same forward and backward}\}$$
- There exists a Turing machine that decides PALINDROMES

# Another example: Primality testing

- **Informal problem statement:** “Given  $K \in \mathbb{N}$ , determine whether  $K$  is prime.”
- **Formulating the problem as a language:**
  - Let  $\langle K \rangle$  denote the binary **encoding** of  $K$ , i.e., the standard base-2 representation of  $K$
  - Example:  $\langle 6 \rangle = 110$ . Note that  $K \in \mathbb{N}$  whereas  $\langle K \rangle \in \{0, 1\}^*$
  - Language:

$$\text{PRIMES} = \{\langle K \rangle : K \text{ is a prime number}\}$$

# Encoding the input as a string

- **OBJECTION:** “Why should I have to **encode** my inputs?”
- **RESPONSE:** Encoding is necessary even for **human** computation!
  - What we say: “Given a nonnegative integer, determine whether it is prime”
  - What we mean: “Given a piece of **text**, determine whether it **represents/encodes** a prime number”



“This is not a pipe.”  
(1929 painting by René Magritte)

# Larger alphabets

A B C

- **OBJECTION:** “Why encode the input in **binary**? Why not other alphabets?”
- **RESPONSE 1:** The Turing machine definition can be modified to handle inputs over other alphabets. We focus on binary inputs **for simplicity’s sake**
- **RESPONSE 2:** We can **encode symbols** from other alphabets in binary

# Example: ASCII

[NUL]	[SOH]	[STX]	[ETX]	[EOT]	[ENQ]	[ACK]	[BEL]	[BS]	[HT]	[LF]	[VT]	[FF]
0000000	0000001	0000010	0000011	0000100	0000101	0000110	0000111	0001000	0001001	0001010	0001011	0001100
[CR]	[SO]	[SI]	[DLE]	[DC1]	[DC2]	[DC3]	[DC4]	[NAK]	[SYN]	[ETB]	[CAN]	[EM]
0001101	0001110	0001111	0010000	0010001	0010010	0010011	0010100	0010101	0010110	0010111	0011000	0011001
[SS]	[ESC]	[FS]	[GS]	[RS]	[US]	[SPACE]	!	"	#	\$	%	&
0011010	0011011	0011100	0011101	0011110	0011111	0100000	0100001	0100010	0100011	0100100	0100101	0100110
'	(	)	*	+	,	-	.	/	0	1	2	3
0100111	0101000	0101001	0101010	0101011	0101100	0101101	0101110	0101111	0110000	0110001	0110010	0110011
4	5	6	7	8	9	:	;	<	=	>	?	@
0110100	0110101	0110110	0110111	0111000	0111001	0111010	0111011	0111100	0111101	0111110	0111111	1000000
A	B	C	D	E	F	G	H	I	J	K	L	M
1000001	1000010	1000011	1000100	1000101	1000110	1000111	1001000	1001001	1001010	1001011	1001100	1001101
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1001110	1001111	1010000	1010001	1010010	1010011	1010100	1010101	1010110	1010111	1011000	1011001	1011010
[	\	]	^	_	`	a	b	c	d	e	f	g
1011011	1011100	1011101	1011110	1011111	1100000	1100001	1100010	1100011	1100100	1100101	1100110	1100111
h	i	j	k	l	m	n	o	p	q	r	s	t
1101000	1101001	1101010	1101011	1101100	1101101	1101110	1101111	1110000	1110001	1110010	1110011	1110100
u	v	w	x	y	z	{		}	~	[DEL]		
1110101	1110110	1110111	1111000	1111001	1111010	1111011	1111100	1111101	1111110	1111111		

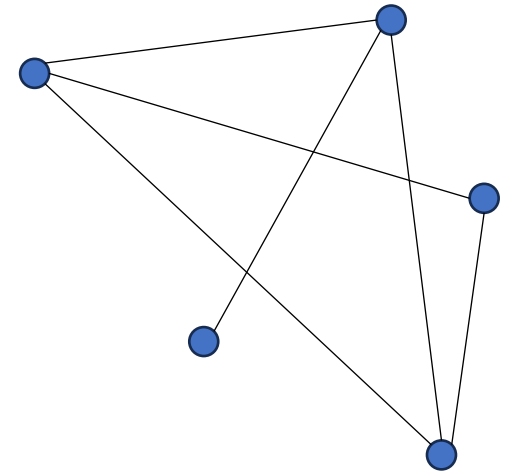
# Another encoding example: Connectivity

- **Informal problem statement:** “Given a  $K$ -vertex graph  $G$ , determine whether it is connected”

- **Formulating the problem as a language:**

- Let  $\langle G \rangle \in \{0, 1\}^{K^2}$  denote the **adjacency matrix** of  $G$
- Language:

$$\text{CONNECTED} = \{\langle G \rangle : G \text{ is a connected graph}\}$$



# Multiple possible encodings

- **OBJECTION:** “Why are we using adjacency **matrices** instead of adjacency **lists**?”
- **RESPONSE:** It doesn’t matter much which encoding we use, because it is not hard to **convert between** the two encodings

# Encoding other things as strings

- If  $X$  is **any mathematical object that can be written down** (a number, a graph, a polynomial, ...), then we use the notation  $\langle X \rangle$  to denote some “reasonable” encoding of  $X$  as a binary string
- It typically doesn’t matter which specific encoding we use, provided we choose something **reasonable**
- If you are unsure how  $\langle X \rangle$  should be defined in a particular case, **ask!**