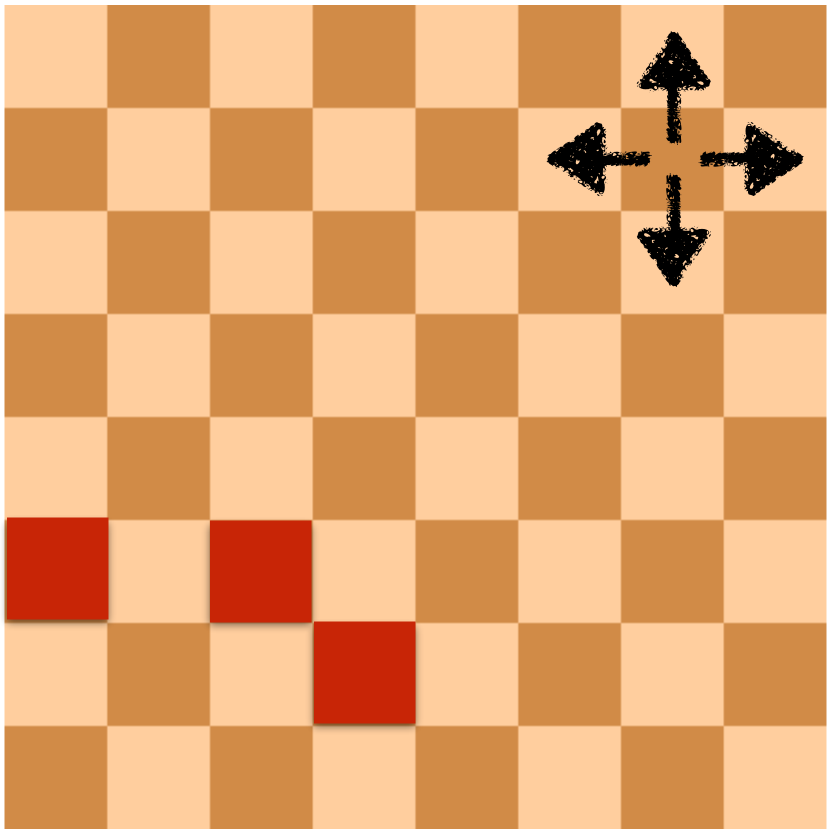
15-251
Great Theoretical Ideas in Computer Science

Lecture 2:
Strings and Encodings

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Jan 19th, 2017

Chessboard Puzzle



neighbors in direction
N, S, W, E

Initially, some of the squares are "infected".

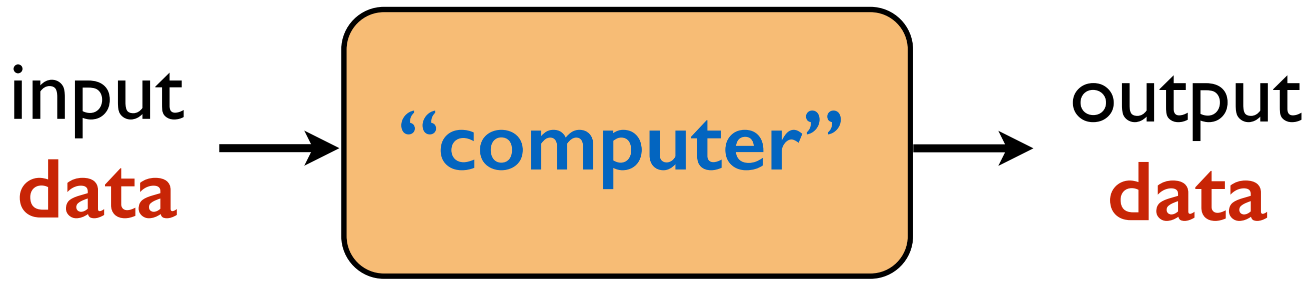
If a square has 2 or more infected neighbors, it becomes infected.

Question: What is the min number of infected squares needed initially to infect the whole board?

Objects/concepts we want to study and understand

Mathematical model (formal, precise definitions)

Mathematically/rigorously prove facts/theorems



Computation: manipulation of **data**.

How do we mathematically/formally represent **data**?

We have already done it for communication purposes.

Written communication:

“apple”

“car”

“happy”

“three” or “3”

English alphabet

$$\Sigma = \{\text{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}\}$$

Turkish alphabet

$$\Sigma = \{\text{a,b,c,ç,d,e,f,g,ğ,h,ı,i,j,k,l,m,n,o,ö,p,r,s,ş,t,u,ü,v,y,z}\}$$

What if we had more symbols?

What if we had less symbols?

Binary alphabet

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

An **alphabet** is a non-empty, finite set (usually denoted by Σ).

An element of an alphabet is called a **symbol** or **character**.

Any (usually finite) sequence of symbols from Σ is called a **string** (or a **word**) over Σ.

A string is denoted by $a_1 a_2 a_3 \ldots a_n$, where each $a_i \in \Sigma$.

Example: Some strings over $\Sigma = \{0, 1\}$:

ϵ $\quad 0 \quad 1 \quad 01 \quad 1011110101101111$

Example: Some strings over $\Sigma = \{a, b, c\}$:

$\epsilon \quad a \quad b \quad c \quad ca \quad caabcccab$

Length of a string s, $|s|$, is the number of symbols in s.

Given an alphabet Σ,

Σ^* denotes the set of all **finite** length strings over Σ.

Examples:

$$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111 \ldots\}$$

$$\{a\}^* = \{\epsilon, a, aa, aaa, aaaa, aaaaa, \ldots\}$$

Written English

$$\Sigma = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z\}$$

<u>Objects/concepts of interest</u> <u>String encoding</u>

apple

car

happy

Does every object have a corresponding encoding?

Can two objects have the same encoding?

Does every string correspond to a valid encoding?

Given a set A of objects, an **encoding** of A is an injective function

$$\mathrm{Enc} : A \to \Sigma^* \ .$$

Notation: For $a \in A$, $\langle a \rangle$ denotes $\mathrm{Enc}(a)$.

Technicality Alert: not all sets are encodable.

Examples

$$A = \mathbb{N}$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$\langle 36 \rangle = \text{``36''}$$

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

$$\langle 36 \rangle = \text{``100100''}$$

$$\Sigma = \{1\}$$

$$\langle 36 \rangle = \text{``111111111111111111111111111111111111''}$$

Does Σ affect "encodability"?

Examples

$$A = \mathbb{Z}$$

$$\Sigma = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$\langle -36 \rangle = \text{“} -36\text{”}$$

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

$$\langle -36 \rangle = \text{“}1100100\text{”}$$

$$\Sigma = \{1\}?$$

Examples

$$A = \mathbb{N} \times \mathbb{N}$$

$$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \#\}$$

$$\langle (3, 36) \rangle = \langle 3, 36 \rangle = \text{“}3\#36\text{”}$$

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

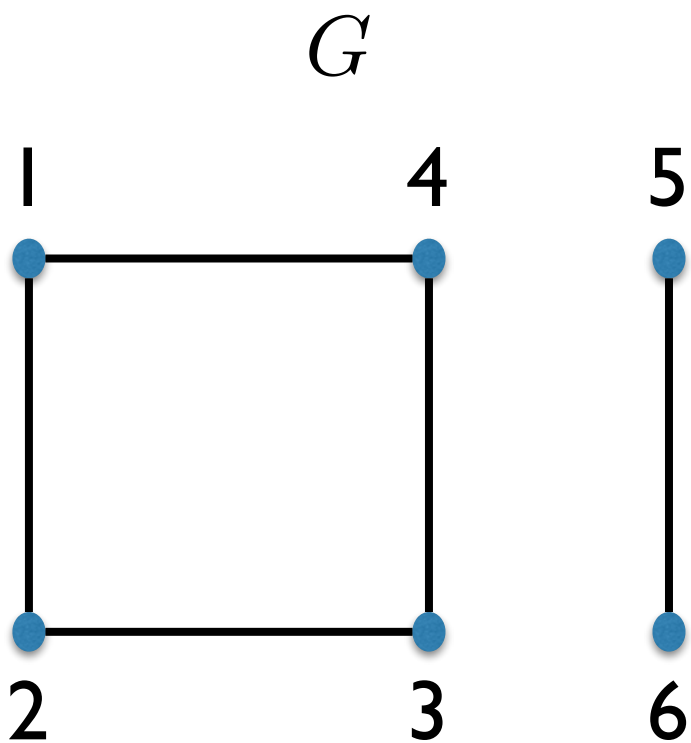
Idea: encode all symbols above using 4 bits (why 4?)

$0 \to 0000$	$4 \to 0100$	$8 \to 1000$
$1 \to 0001$	$5 \to 0101$	$9 \to 1001$
$2 \to 0010$	$6 \to 0110$	$\# \to 1010$
$3 \to 0011$	$7 \to 0111$	

$$\langle 3, 36 \rangle = \text{“}0011101000110110\text{”}$$

Examples

$A =$ all undirected graphs

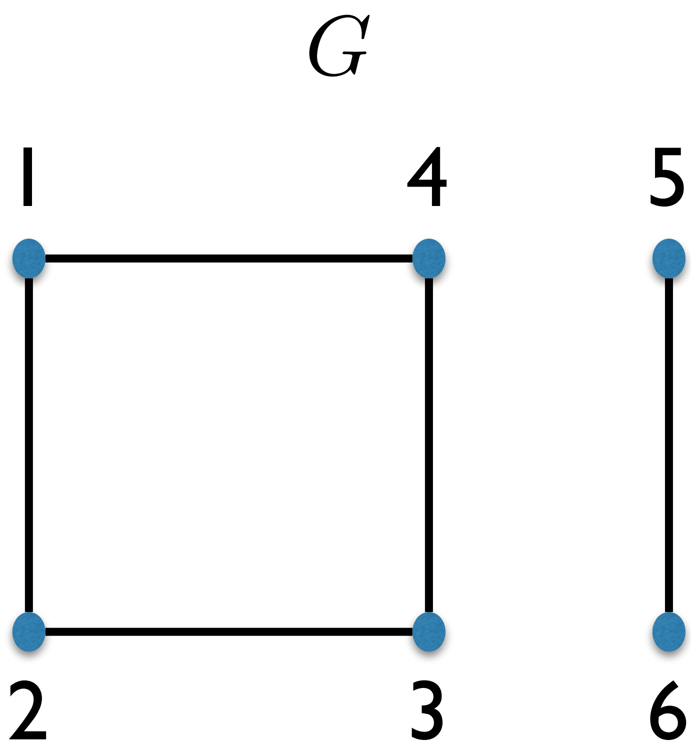


$\langle G \rangle =$

" V = {1, 2, 3, 4, 5, 6}
E = {{1,2}, {2,3}, {3,4}, {1,4}, {5,6}}"

Examples

$A =$ all undirected graphs



$$\begin{array}{c c} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

$\langle G \rangle =$

"$010100\#101000\#010100\#101000\#000001\#000010$"

Examples

$A =$ all Python functions

```
def isPrime(N):
    if (N < 2):
        return False
    for factor in range(2, N):
        if (N % factor == 0):
            return False
    return True
```

$\langle \text{isPrime} \rangle =$

"def isPrime(N):\n   if (N < 2):\n   return False\n   for factor in range(2, N):\n   if (N % factor == 0):\n   return False\n   return True"

Does $|\Sigma|$ matter?

Going from $|\Sigma| = k$ to $|\Sigma'| = \boxed{2}$:

encode every symbol of Σ using t bits,

where $t = \lceil \log_{\boxed{2}} k \rceil$.

A word of length n over Σ $\longrightarrow$ A word of length tn over Σ'

Does $|\Sigma|$ matter?

	Binary vs	Unary
0	0	ϵ
1	1	1
2	10	11
3	11	111
4	100	1111
5	101	11111
6	110	111111
7	111	1111111
8	1000	11111111
9	1001	111111111
10	1010	1111111111
11	1011	11111111111
12	1100	111111111111

Does $|\Sigma|$ matter?

Binary vs Unary

n **has length** $\lfloor \log_2 n \rfloor + 1$ **in binary**

n **has length** n **in unary**

n **has length** $\lfloor \log_k n \rfloor + 1$ **in base** k

Unary is exponentially longer than other bases!

Which sets are encodable?

Encodability = Countability

(Lecture 7)

What about uncountable sets?

Approximate.

Data is represented as finite length strings over some finite alphabet.

Reasoning about computation requires reasoning about strings.

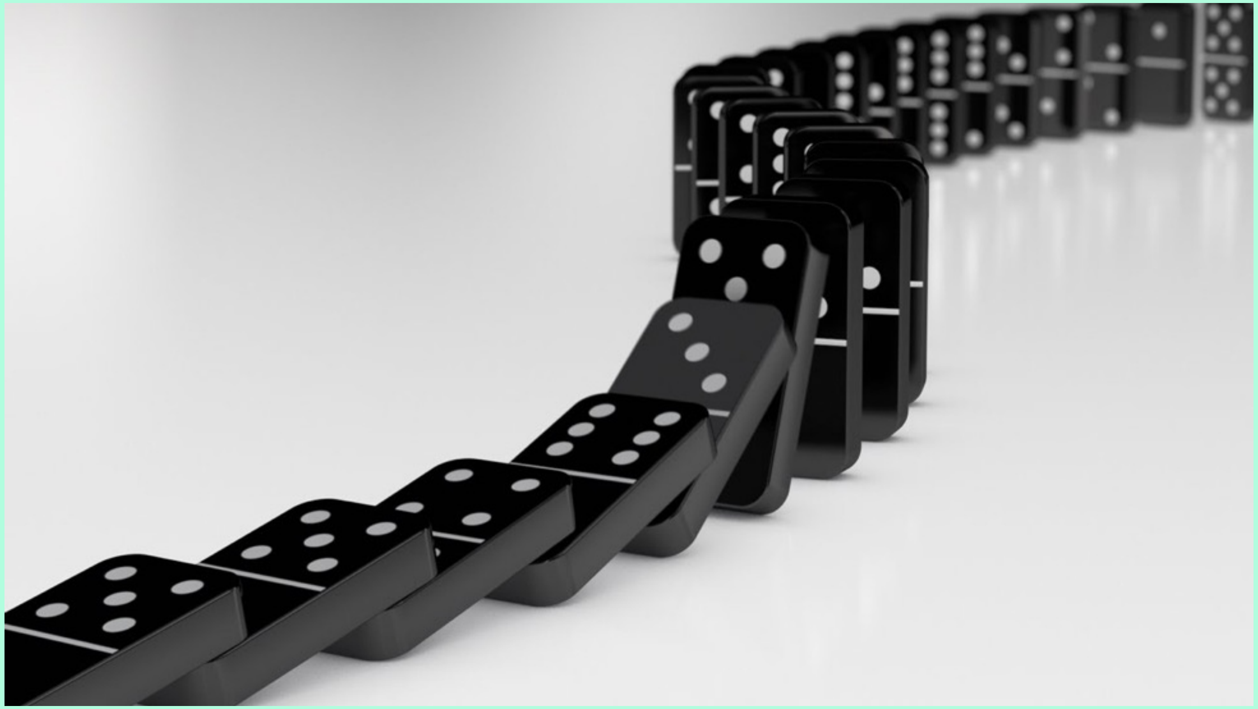
Inductive Reasoning

(powerful tool for understanding recursive structures)

Induction Review

<u>Domino Principle</u>

Line up any number of dominos in a row, knock the first one over and they will all fall.



Induction Review

<u>Domino Principle</u>

Line up an infinite row of dominoes,
one domino for each natural number.
Knock the first one over and they will all fall.

<u>Proof</u>: Proof by contradiction: suppose they don't all fall.
Let **k** be the *lowest numbered domino* that remains standing.
Domino **k-1** did fall. But then **k-1** knocks over **k**, and **k** falls.
So **k** stands and falls, which is a contradiction.

Induction Review

<u>Mathematical induction:</u>

statements proved instead of dominoes fallen

Infinite sequence of dominoes

F_k = "domino k fell"

Infinite sequence of statements: $S_0, S_1, S_2, \ldots$

F_k = "S_k proved"

<u>Establish:</u>
1. F_0
2. for all k, $F_k \Longrightarrow F_{k+1}$

<u>Conclude:</u> F_k is true for all k.

Induction Review

<u>Mathematical induction:</u>

statements proved instead of dominoes fallen

Infinite sequence of dominoes

Infinite sequence of statements: $S_0, S_1, S_2, \ldots$

F_k = "domino k fell"

F_k = "S_k proved"

"Strong" Induction

<u>Establish:</u>

1. F_0
2. for all k, $F_0, F_1, \ldots, F_k \Longrightarrow F_{k+1}$

<u>Conclude:</u> F_k is true for all k.

Different ways of packaging inductive reasoning

"Method of Min Counterexample"

Example:

Every natural number > 1 can be factored into primes.

Proof (by contradiction):

Let n be the smallest counter-example.

n cannot be prime, so $n = ab$, where $1 < a, b < n$.

Since n is the smallest counter-example,
a and b must have prime factorizations.

Then so does n. Contradiction.

Different ways of packaging induction proofs

"Method of Min Counterexample"

<u>The general idea of method of min counterexample:</u>

By contradiction.

Let k be the **min** number such that S_k is not true.

Show that $S_{k'}$ is not true for k' < k. Contradiction.

Different ways of packaging induction proofs

"Invariant Induction"

Example:

At any party, at any point in time,
define a person's **parity** as **odd**/**even** according to the number of hands they have shaken.

Statement: number of people of **odd** parity must be even.

Different ways of packaging induction proofs

"Invariant Induction"

Statement: number of people of **odd** parity must be even.

Proof:

Initial state:

0 hands have been shaken. 0 people have **odd** parity.

Invariant argument:

At an arbitrary point in the party,
let **t** be the number # people with **odd** parity.

odd	**odd**		**t <— t-2**
even	**even**		**t <— t+2**
odd	**even**		**t <— t**
even	**odd**		**t <— t**

parity of **t** doen't change.

Different ways of packaging induction proofs

"Invariant Induction"

<u>The general idea of invariant induction:</u>

Time-varying world state: $W_0, W_1, W_2, \ldots$

<u>Want to prove</u>: statement S is true for all world states.

<u>Argue</u>:

Statement S is true for W_0.

If S is true for W_k, it remains true for W_{k+1}.

Different ways of packaging induction proofs

"Structural Induction"

Induction on objects with a recursive structure.

- arrays/lists
- strings
- graphs

⋮

Different ways of packaging induction proofs

"Structural Induction"

Recursive definition of a **string** over Σ :

- the empty sequence ϵ is a string.
- if x is a string and $a \in \Sigma$, then ax is a string.

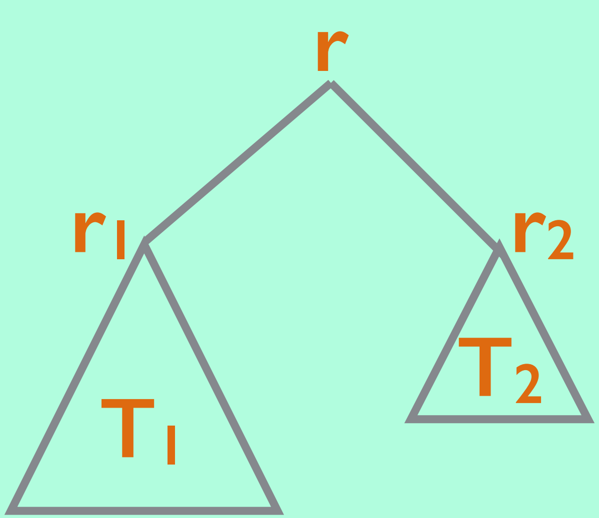
Different ways of packaging induction proofs

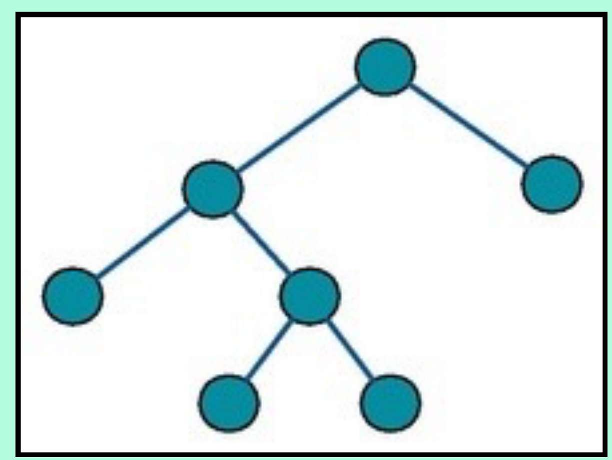
"Structural Induction"

Recursive definition of a **rooted binary tree**:

- a single node r is a binary tree with root r.
- if T_1 and T_2 are binary trees with roots r_1 and r_2, then T which has a node r adjacent to r_1 and r_2 is a binary tree with root r.

$T =$





Every node has 0 or 2 children.

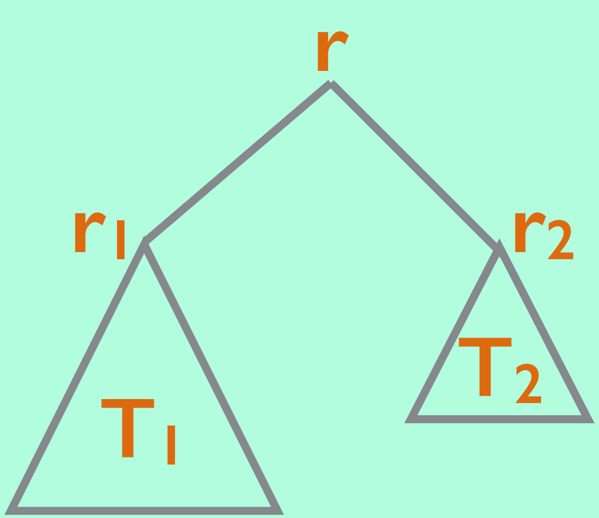
Different ways of packaging induction proofs

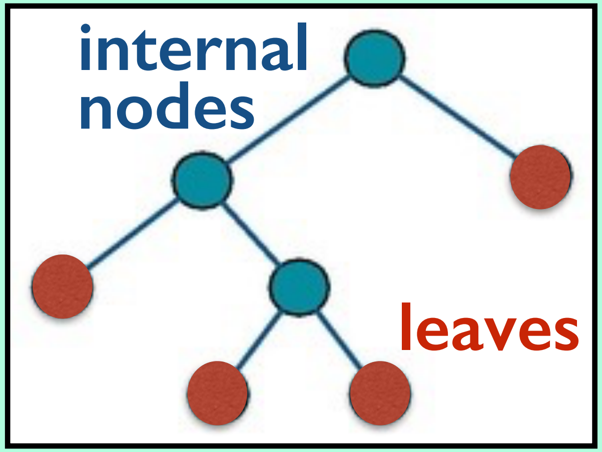
"Structural Induction"

Recursive definition of a **rooted binary tree**:

- a single node r is a binary tree with root r.
- if T_1 and T_2 are binary trees with roots r_1 and r_2, then T which has a node r adjacent to r_1 and r_2 is a binary tree with root r.

$T =$





Every node has 0 or 2 children.

Different ways of packaging induction proofs

"Structural Induction"

Example: Let T be a binary tree.

Let L_T = # leaves in T.

Let I_T = # internal nodes in T.

Then $L_T = I_T + 1$.

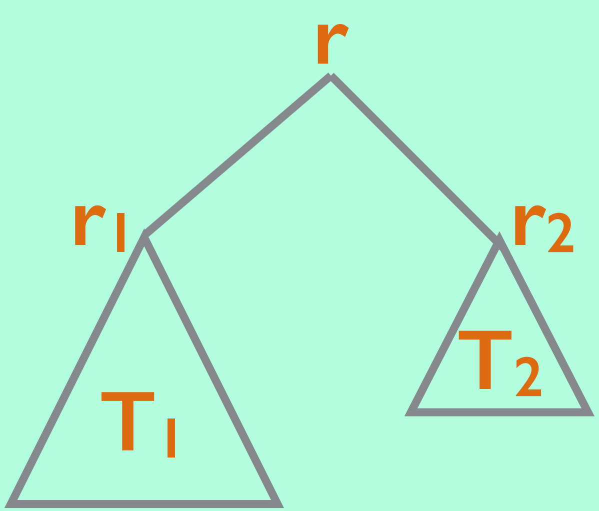
Different ways of packaging induction proofs

"Structural Induction"

<u>Proof (by structural induction)</u>:

Base case (T is a single node) is true.

Let T be an arbitrary binary tree:

$T =$ 

We know $L_T = L_{T1} + L_{T2}$

and $I_T = I_{T1} + I_{T2} + 1$.

By IH: $L_{T1} = I_{T1} + 1$ and $L_{T2} = I_{T2} + 1$.

So $L_T = L_{T1} + L_{T2} = I_{T1} + 1 + I_{T2} + 1 = I_T + 1$.

Different ways of packaging induction proofs

"Structural Induction"

<u>The general idea of structural induction:</u>

<u>Base step</u>: check statement true for base case(s) of def'n.

<u>Recursive/induction step</u>:
prove statement holds for **new objects** created by the recursive rule, assuming it holds for **old objects** used in the recursive rule.

Different ways of packaging induction proofs

"Structural Induction"

Why is that valid?

Follows from strong induction on # of applications of the recursive rule to create a particular object.

(even though we don't phrase it explicitly that way)

Previous example: Could have also packaged it as strong induction on the parameter **height**.

Different ways of packaging induction proofs

"Structural Induction"

Be careful! What is wrong with the following argument?

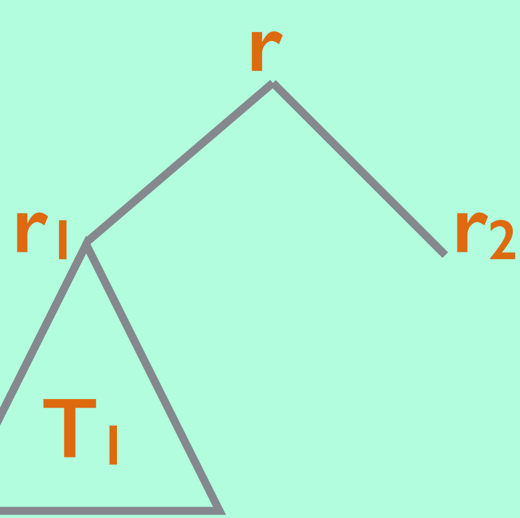
Strong induction on height.

Base case true.

Take an arbitrary binary tree **T** of height **h**.

Let **T'** be the following tree of height **h+1**:

T' =



blah blah blah

Therefore statement true for **T'** of height **h+1**.

Different ways of packaging induction proofs

"Structural Induction"

Another example with strings:

Let $L \subseteq \{0,1\}^*$ be recursively defined as follows:

- $\epsilon \in L$;
- if $x, y \in L$, then $0x1y0 \in L$.

Prove that for any $w \in L$, $\#(0, w) = 2 \cdot \#(1, w)$.

number of 0's in w

number of 1's in w

Different ways of packaging induction proofs

"Structural Induction"

<u>Proof (by structural induction):</u>

Base case is $w = \epsilon$ and $\#(0, \epsilon) = 2 \cdot \#(1, \epsilon)$.

Assume statement is true for all $u \in L, |u| < k$.

Let w be an arbitrary element of L with $|w| = k$.

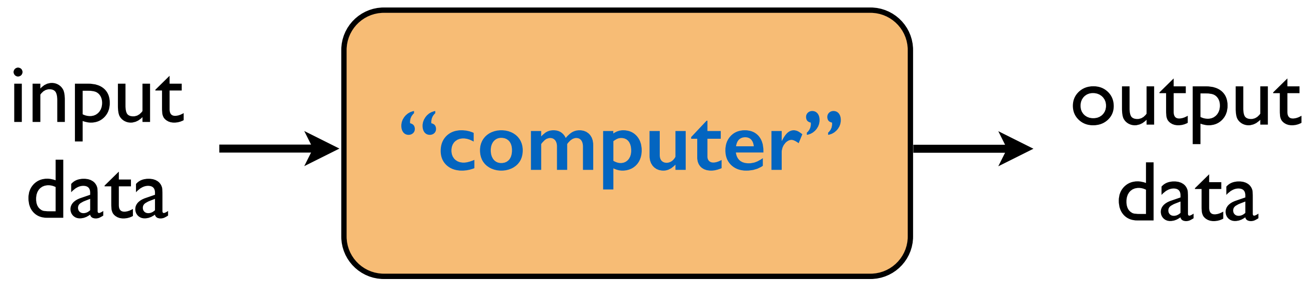
So $w = 0x1y0$ for some $x, y \in L,\ |x| < k,\ |y| < k$.

By IH: $\#(0, x) = 2 \cdot \#(1, x)$ and $\#(0, y) = 2 \cdot \#(1, y)$.

Then:
$$\begin{aligned}\#(0, w) &= 2 + \#(0, x) + \#(0, y) \\ &= 2 + 2 \cdot \#(1, x) + 2 \cdot \#(1, y) \\ &= 2(1 + \#(1, x) + \#(1, y)) = 2 \cdot \#(1, w)\end{aligned}$$

Back to string encodings

First Few Weeks



What is computation?

What is an algorithm?

How can we mathematically define them?

Seen so far:

Can encode/represent any kind of data
(*numbers*, *text*, *pairs of numbers*, *graphs*, *images*, *etc…*)
with a **finite length (binary) string**.

Before we define **algorithm** formally,
we should define **computational problem** formally.

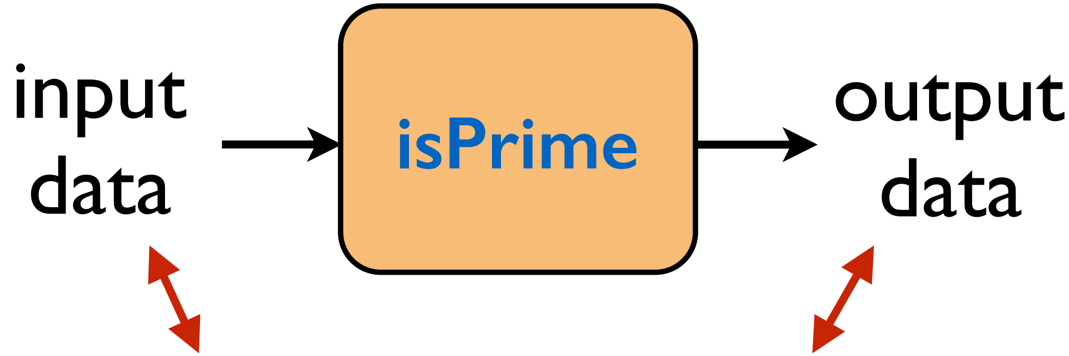
An **algorithm** *solves* a **computational problem**.

Example description of a computational problem:

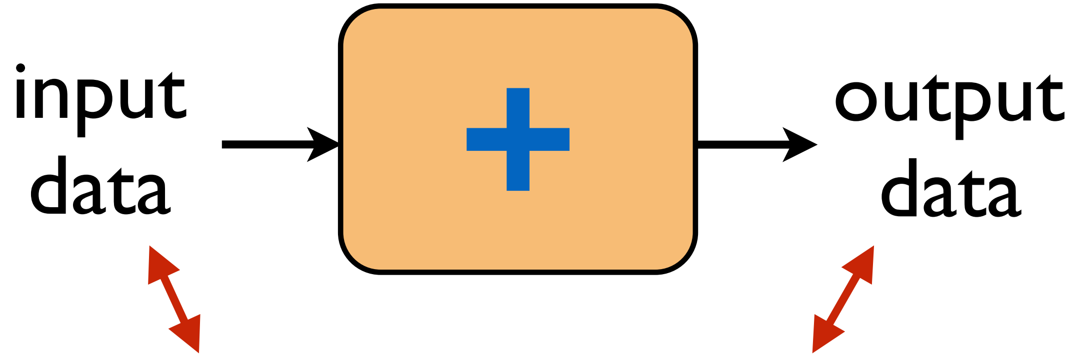
Given a natural number **N**, output *True* if **N** is prime, and output *False* otherwise.

Example algorithm *solving* it:

```
def isPrime(N):
    if (N < 2): return False
    for factor in range(2, N):
        if (N % factor == 0): return False
    return True
```



Instance	Solution
0	No
1	No
2	Yes
3	Yes
4	No
$\vdots$	$\vdots$
251	Yes
$\vdots$	$\vdots$



Instance	Solution
0, 0	0
0, 1	1
1, 1	2
2, 2	4
2, 3	5
10, 1	11
100, 99	199
$\vdots$	$\vdots$

Instance

["vanilla", "mind", "Anil", "yogurt", "doesn't"]

Solution

["Anil", "doesn't", "mind", "vanilla", "yogurt"]

A computational problem is a function

$$f : A \to B \, .$$

$A =$ set of possible input objects (called instances)

$B =$ set of possible output objects (called solutions)

But in TCS, we don't deal with arbitrary objects,
we deal with strings (encodings).

$$\boxed{\begin{array}{ccccc} f : & A & \to & B \\ & \downarrow & \text{Enc} & \downarrow \\ f' : & \Sigma^* & \to & \Sigma^* \end{array}}$$

Technicality:

What if $w \in \Sigma^*$ does not correspond to an encoding of an instance?

IMPORTANT DEFINITIONS

Definition: A *computational problem* is a function

$$f : \Sigma^* \to \Sigma^*.$$

Definition: A *decision problem* is a function

$$f : \Sigma^* \to \{0, 1\}.$$

No, Yes

False, True

Reject, Accept

Definition: A subset $L \subseteq \Sigma^*$ is called a *language*.

IMPORTANT RELATIONSHIP

There is a one-to-one correspondence between decision problems and languages.

Instance	Solution
$\boxed{\epsilon}$	1
$\boxed{0}$	1
$\boxed{1}$	1
$\boxed{00}$	1
01	0
10	0
$\boxed{11}$	1
$\boxed{000}$	1
001	0
$\vdots$	$\vdots$

$$L \subseteq \Sigma^*$$

$$L = \{\epsilon, 0, 1, 00, 11, 000, \ldots\}$$

Our focus will be on languages!
(decision problems)

- Convenient restriction.
- Usually "without loss of generality".
  (more on this next lecture)

INTERESTING QUESTIONS WE WILL EXPLORE ABOUT COMPUTATION

Are all languages computable/decidable?

How can we prove that a language is not decidable?

How do we measure complexity of algorithms deciding languages?

How do we classify languages according to resources needed to decide them?

P = NP?