

Unit 5

Unit 5

Undecidability

Properties of
Recursive and
Recursively
enumerable
languages

Introduction to Computational Complexity

Decidability vs
Undecidability

Examples of
Undecidable
Problem

Rice Theorem

Post
Correspondence
Problem

Undecidable
problems about
Turing Machine-
Post's
Correspondence
Problem

Time and Space
complexity of
TMs

Complexity
classes: Class P,
Class NP

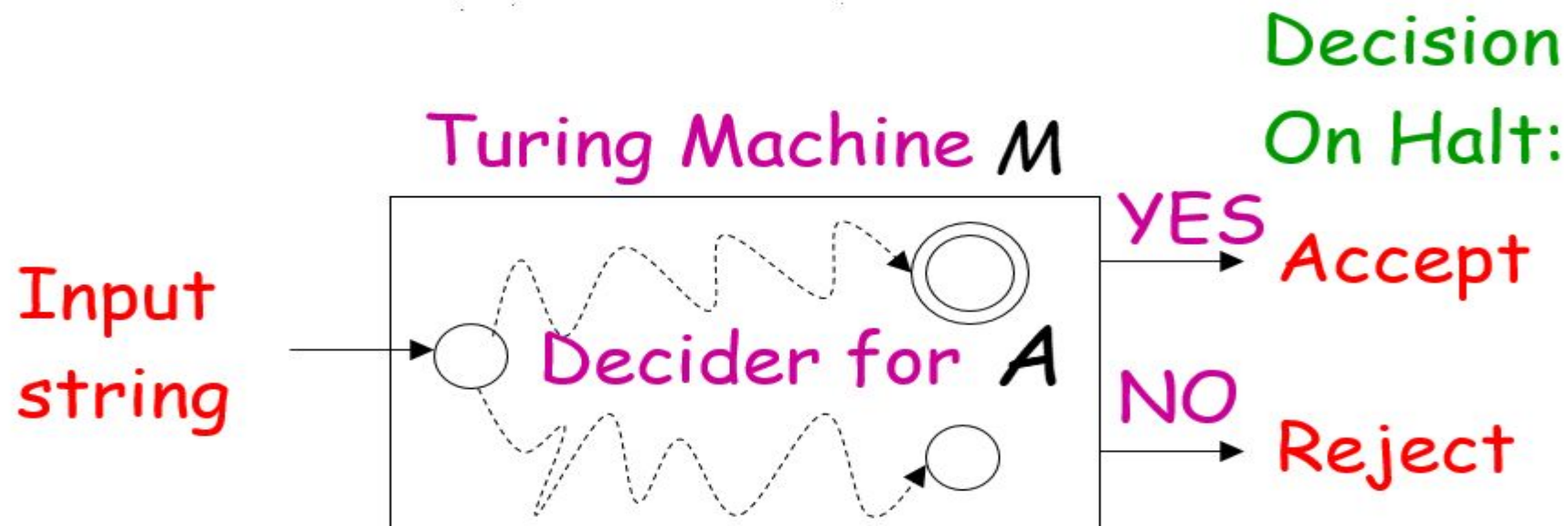
NP hardness

NP Completeness

Undecidability

Decidable Languages

- A language is **decidable**,
- if there is a Turing machine (**decider**)
- that accepts the language and
- halts on every input string



Decidable Problem

- If there is a Turing machine that decides the problem, called as Decidable problem.
- A decision problem that can be solved by an algorithm that halts on all inputs in a finite number of steps.
- A problem is decidable, if there is an algorithm that can answer either yes or no.
- A language for which membership can be decided by an algorithm that halts on all inputs in a finite number of steps.
- Decidable problem is also called as totally decidable problem, algorithmically solvable, recursively solvable.

Decidability

Question: Why study decidability?

- Your boss orders you to solve Hilbert's 10th, or else
- Good for your imagination.
- Some of the most beautiful and important mathematics of the 20th century, and you can actually understand it! (YMMV).

Example

Finite automata problems can be reformulated as languages.

Does DFA B accept input string w ?

Consider the language:

$$A_{\text{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts } w\}$$

These problems are equivalent:

- B accepts w
- $\langle B, w \rangle \in A_{\text{DFA}}$

Theorem

Theorem: A_{DFA} is a decidable language.

On input $\langle B, w \rangle$, where B is a DFA and w a string:

1. Simulate B on input w
2. if simulation ends in accepting state, *accept*,
otherwise *reject*.

Note

- “where” clause means scan and check condition.
- B represented by a list of $(Q, \Sigma, \delta, q_0, F)$.
- simulation straightforward

Theorem

Does an NFA accept a string?

$$A_{\text{NFA}} = \{\langle B, w \rangle \mid B \text{ is an NFA that accepts } w\}$$

Theorem: A_{NFA} is a decidable language.

On input $\langle B, w \rangle$, where B is an NFA and w a string:

1. Convert. NFA B into equivalent C
2. Run previous TM on input $\langle C, w \rangle$.
3. if that TM accepts, accept, otherwise reject.

Note use of subroutine.

Theorem

Does a regular expression generate a string?

$$A_{\text{REX}} = \{\langle R, w \rangle \mid R \text{ is a regular expression that generates } w\}$$

Theorem: A_{REX} is a decidable language.

On input $\langle R, w \rangle$, where R is a regular expression and w a string:

1. Convert regular expression R into equivalent C .
2. Run earlier TM on input $\langle C, w \rangle$.
3. If that TM accepts, accept, otherwise reject.

Theorem

Does a DFA accept the empty language?

$$E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Theorem: E_{DFA} is a decidable language.

On input $\langle A \rangle$, where A is a DFA:

Mark the start state of A .

2. Repeat until no new states are marked:
3. Mark any state that has a transition coming into it from any already marked state.
4. If no accept state is marked, accept, otherwise reject.

This TM actually just tests whether any accepting state is reachable from initial state.

Undecidable Languages

- undecidable language = not decidable language
- There is no decider:
 - there is no Turing Machine which accepts the language and makes a decision (halts) for every input string
 - (machine may make decision for some input strings)
 - there is no Turing Machine (Algorithm) that gives an answer (yes or no) for every input instance
 - (answer may be given for some input instances)

Undecidable problem

- A problem that cannot be solved for all cases by any algorithm whatsoever.
- Equivalent Language cannot be recognized by a Turing machine that halts for all inputs.

The following problems are undecidable problems:

- Halting Problem: A halting problem is undecidable problem. There is no general method or algorithm which can solve the halting problem for all possible inputs.
- Emptiness Problem: Whether a given TM accepts Empty?
- Finiteness Problem: Whether a given TM accepts Finite?
- Equivalence Problem: Whether Given two TM's produce same language?. Is $L(TM1) = L(TM2)$?
- Is $L(TM1) \subseteq L(TM2)$? (Subset Problem)
- Is $L(TM1) \cap L(TM2) = CFL$?
- Is $L(TM1) = \Sigma^*$? (Totality Problem)
- Is the complement of $L(G1)$ context-free ?

Halting Problem

One of the most philosophically important theorems of the theory of computation.

Computers (and computation) are limited in a very fundamental way.

Common, every-day problems are unsolvable.

- does a program sort an array of integers?
- both program and specification are precise mathematical objects.
- proving program \cong specification should be just like proving that triangle 1 \cong triangle 2 ...

Does a Turing machine accept a string?

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts } w\}$$

Theorem: A_{TM} is undecidable.

Recall that A_{DFA} , A_{NFA} , and A_{CFG} are decidable.

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts } w\}$$

But, first note that

Theorem: A_{TM} is enumerable.

Define the machine U .

On input $\langle M, w \rangle$, where M is a TM and w a string

1. Simulate M on input w .
2. If M ever enters its accept state, *accept*, and if M ever enters its reject state, *reject*.

Turing Machines are countable

Claim: The set of strings in Σ^* is countable.

Proof: List strings of length 1, then length 2, and so on.

Claim: The set of Turing machines is countable.

Proof: Each TM M has an encoding as a string $\langle M \rangle$.

We can list all strings, and just omit the ones that are not legal TM encodings.

Recap : Turing machines and computability

1. Definition of Turing machines: high level and low-level descriptions
2. Variants of Turing machines
3. Decidable and Turing recognizable languages
4. Church-Turing Hypothesis UNDECIDABILITY
5. Undecidability and a proof technique by diagonalization

A universal TM lang $L_{TM}^A = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

Undecidable problems

- L is empty?
- L is regular?
- L has size 2?

This can be generalized to all non-trivial properties of Turing-acceptable languages

Rice's theorem (1953)

1. Any non-trivial property of R.E languages is undecidable!
2. Property $P \equiv$ set of languages satisfying P
3. Property of r.e languages: membership of M in P depends only on the language of M .

If $L(M) = L(M')$, then $\langle M \rangle \in P$ if $\langle M' \rangle \in P$

4. Non-trivial: It holds for some but not all TMs.

Properties of RE Languages

- Let $RE = \{\mathcal{L}(M) \mid M \text{ is a Turing machine}\}$.
- RE is the class of all r.e. languages.
- A property of r.e. sets is a map

$$P : RE \rightarrow \{T, F\}.$$

- Example: Emptiness is a property defined as

$$P_{EMP}(L) = \begin{cases} T & \text{if } L = \emptyset \\ F & \text{if } L \neq \emptyset \end{cases}$$

- R.E. languages are specified by Turing machines.
- Properties too are specified by Turing machines.
- Example: The emptiness property is specified by **any member** of

$$P_{EMP} = \{M \mid \mathcal{L}(M) = \emptyset\}.$$

Property : Set of language

Just a subset of r.e. languages. Thus, **L satisfies a property P** if $L \in P$.

Examples:

1. Set of regular languages \mathcal{R}
2. Set of context-free languages \mathcal{CFL}
3. Set of all languages \mathcal{L}
4. $\{\emptyset\}$
5. \emptyset

Given a property P , denote $\mathcal{L}_P = \{\langle M \rangle \mid L(M) \in P\}$

- So decidability of property P is decidability of language \mathcal{L}_P .

Type of Property

- **Trivial properties**

- The constant map $\text{RE} \rightarrow \{T, F\}$ taking all $L \in \text{RE}$ to T .
- The constant map $\text{RE} \rightarrow \{T, F\}$ taking all $L \in \text{RE}$ to F .
- Any other property is called **non-trivial**.
- Example of trivial property: $\mathcal{L}(M)$ is recursively enumerable.
- Example of non-trivial property: $\mathcal{L}(M)$ is recursive.

- **Monotone properties**

- Assume $F \leq T$.
- Whenever $A \subseteq B$, we have $P(A) \leq P(B)$.
- Examples of monotone properties: $\mathcal{L}(M)$ is infinite, $\mathcal{L}(M) = \Sigma^*$.
- Examples of non-monotone properties: $\mathcal{L}(M)$ is finite, $\mathcal{L}(M) = \emptyset$.

Non-trivial property

For a property P , $\mathcal{L}_P = \{\langle M \rangle \mid L(M) \in P\}$.

A property P of r.e. languages is called *non-trivial* if

- ▶ $\mathcal{L}_P \neq \emptyset$, i.e., there exists TM M , $L(M) \in P$.
- ▶ $\mathcal{L}_P \neq \text{all r.e. languages}$, i.e., there exists TM M , $L(M) \notin P$.

Non-trivial property:

A property P possessed by some Turing-acceptable languages but not all

Example: $P_1 : L$ is empty?

YES $L = \emptyset$

NO $L = \{\text{Louisiana}\}$

NO $L = \{\text{Baton, Rouge}\}$

More examples of non-trivial properties

P_2 : L is regular?

YES $L = \emptyset$

YES $L = \{a^n : n \geq 0\}$

NO $L = \{a^n b^n : n \geq 0\}$

P_3 : L has size 2?

NO $L = \emptyset$

NO $L = \{\text{Louisiana}\}$

YES $L = \{\text{Baton Rouge}\}$

Trivial property:

A property P possessed by ALL
Turing-acceptable languages

Examples: P_4 : L has size at least 0?
True for all languages

P_5 : L is accepted by some
Turing machine?

True for all
Turing-acceptable languages

We can describe a property P as the set of languages that possess the property

If language L has property P then $L \in P$

Example: $P : L$ is empty?

YES $L_1 = \emptyset$

$P = \{L_1\}$

NO $L_2 = \{\text{Louisiana}\}$

NO $L_3 = \{\text{Baton, Rouge}\}$

Example: Suppose alphabet is $\Sigma = \{a\}$

P : L has size 1?

NO \emptyset

YES $\{\lambda\}$ $\{a\}$ $\{aa\}$ $\{aaaa\}$ | |

NO $\{\lambda, a\}$ $\{\lambda, aa\}$ $\{a, aa\}$ | |

NO $\{\lambda, a, aa\}$ $\{aa, aaa, aaaa\}$ | |

$P = \{\{\lambda\}, \{a\}, \{aa\}, \{aaaa\}, \{aaaaa\}, \square\}$

Non-trivial property problem

Input: Turing Machine M

Question: Does $L(M)$ have the non-trivial
property P ? $L(M) \in P$?

Corresponding language:

$PROPERTY_{TM} = \{\langle M \rangle : M \text{ is a Turing machine}$
such that $L(M)$ has the non - trivial
property P , that is, $L(M) \in P\}$

Can u apply rice theorem ?

1. $\{\langle M \rangle \mid M \text{ runs for 5 steps on word } 010\}$.
2. $\{\langle M \rangle \mid L(M) \text{ is recognized by a TM with at least 25 states.}\}$.
3. $\{\langle M \rangle \mid L(M) \text{ is recognized by a TM with at most 25 states.}\}$.
4. $\{\langle M \rangle \mid M \text{ has at most 25 states.}\}$.
5. $\{\langle M \rangle \mid L(M) \text{ is finite.}\}$.
6. $\{\langle M \rangle \mid M \text{ with alphabet } \{0, 1, \sqcup\} \text{ ever prints three consecutive 1's on the tape}\}$.

For each of No answers above, is the language decidable?

What do you do when Rice's theorem does not apply? Fall back on reductions!

Rice Theorem (Part 1)

Theorem

Any **non-trivial** property P of r.e. languages is undecidable. In other words, the set $\Pi = \{N \mid P(\mathcal{L}(N)) = T\}$ is not recursive.

Proof

- Let P be a non-trivial property of r.e. languages.
- Suppose $P(\emptyset) = F$ (the other case can be analogously handled).
- Since P is non-trivial, there exist $L \in \text{RE}$, $L \neq \emptyset$, such that $P(L) = T$.
- Let K be a Turing machine with $\mathcal{L}(K) = L$.
- We make a reduction from HP to Π .

Rice Theorem – The reduction

- **Input:** $M \# w$ (an instance of HP)
 - **Output:** A Turing machine N such that $P(\mathcal{L}(N)) = T$ if and only if M halts on w .
 - Behavior of N on input v :
 - Copy v to a separate tape.
 - Write w to the first tape, and simulate M on w .
 - If the simulation halts:
 - Simulate K on v .
 - Accept if and only if K accepts v .
-
- If M halts on w , $\mathcal{L}(N) = \mathcal{L}(K) = L$.
 - If M does not halt on w , $\mathcal{L}(N) = \emptyset$.
 - $P(L) = T$ and $P(\emptyset) = F$.

Rice Theorem (Part 2)

Theorem

*No **non-monotone** property P of r.e. languages is semidecidable. In other words, the set $\Pi = \{N \mid P(\mathcal{L}(N)) = T\}$ is not recursively enumerable.*

Proof

- P is non-monotone. So there exist r.e. languages L_1 and L_2 such that
$$L_1 \subseteq L_2, \quad P(L_1) = T, \quad P(L_2) = F.$$
- Take Turing machines M_1, M_2 such that $\mathcal{L}(M_1) = L_1$ and $\mathcal{L}(M_2) = L_2$.
- We supply a reduction from $\overline{\text{HP}}$ to Π .
- The reduction algorithm embeds the information of M, w, M_1 , and M_2 in the finite control of N .

Rice Theorem (Part 2) – The reduction

- **Input:** $M \# w$.
 - **Output:** A Turing machine N such that $P(\mathcal{L}(N)) = T$ if and only if M does **not** halt on w .
 - Behavior of N on input v :
 - Copy v from the first tape to the second tape, and w from the finite control to the third tape.
 - Run three simulations in parallel (one step of each in round-robin fashion)
 - M_1 on v on the first tape,
 - M_2 on v on the second tape,
 - M on w on the third tape.
 - Accept if and only if one of the following conditions hold:
 - (1) M_1 accepts v ,
 - (2) M halts on w , and M_2 accepts v .
-
- M does not halt on $w \Rightarrow N$ accepts by (1) $\Rightarrow \mathcal{L}(N) = \mathcal{L}(M_1) = L_1$.
 - If M halts on w , N accepts if either M_1 or M_2 accepts v . In this case, $\mathcal{L}(N) = \mathcal{L}(M_1) \cup \mathcal{L}(M_2) = L_1 \cup L_2 = L_2$ (since $L_1 \subseteq L_2$).

POST'S CORRESPONDENCE PROBLEM

Post Correspondence Problem

- The Post Correspondence Problem (PCP), introduced by Emil Post in 1946, is an undecidable decision problem.
- The PCP problem over an alphabet Σ is stated as follows –
- Given the following two lists, **M** and **N** of non-empty strings over Σ –
- $M = (x_1, x_2, x_3, \dots, x_n)$
- $N = (y_1, y_2, y_3, \dots, y_n)$
- We can say that there is a Post Correspondence Solution, if for some i_1, i_2, \dots, i_k , where $1 \leq i_j \leq n$, the condition $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$ satisfies.

Example 1

Find whether the lists

$M = (abb, aa, aaa)$ and $N = (bba, aaa, aa)$

have a Post Correspondence Solution?

Solution for Example 1

	x_1	x_2	x_3
M	Abb	aa	aaa
N	Bba	aaa	aa

Here,

$$\mathbf{x_2x_1x_3 = 'aaabbbaaa'}$$

$$\text{and } \mathbf{y_2y_1y_3 = 'aaabbbaaa'}$$

We can see that

$$\mathbf{x_2x_1x_3 = y_2y_1y_3}$$

Hence, the solution is $\mathbf{i = 2, j = 1, \text{ and } k = 3.}$

Example 2

- Find whether the lists **M = (ab, bab, bbaaa)** and **N = (a, ba, bab)** have a Post Correspondence Solution?

Solution for Example 2

	x_1	x_2	x_3
M	ab	bab	bbaaa
N	a	ba	bab

In this case, there is no solution because –

$$|x_2x_1x_3| \neq |y_2y_1y_3| \quad (\text{Lengths are not same})$$

Hence, it can be said that this Post Correspondence Problem is **undecidable**.

- **General Objective:**
 - to understand the concept of PCP
- **Specific Objectives**
 - State the objective of PCP
 - Define PCP problem
 - Check whether the PCP instance have a solution or not.
 - Define MPCP
 - Illustrate the conversion of TM to MPCP with an example

Post Correspondence Problem

- It involves with strings
- Goal:
 - To prove this problem about strings to be undecidable

Post Correspondence Problem

- FORMAL DEFINITION

□ Given two lists of strings A and B (equal length)

$$A = w_1, w_2, \dots, w_k \quad B = x_1, x_2, \dots, x_k$$

The problem is to determine if there is a sequence of integers i_1, i_2, \dots, i_m such that:

$$w_{i_1} w_{i_2} \dots w_{i_m} = x_{i_1} x_{i_2} \dots x_{i_m}$$

Example

	A	B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

This PCP instance has a solution: 2, 1, 1, 3:

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 101111110$$

Does this PCP instance have a solution?

	A	B
i	w_i	x_i
1	110	110110
2	0011	00
3	0110	110

~~This PCP instance has a solution: 2,3,1~~
 ~~$w_2 w_3 w_1 = x_2 x_3 x_1 = 00110110110$~~

One more solution:

2,1,1,3,2,1,1,3

Modified Post Correspondence Problem (MPCP)

Definition:

- first pair in the A and B lists must be the first pair in the solution, i.e., the problem is to determine if there is a sequence of zero or more integers i_1, i_2, \dots, i_m such that:

$$w_1 w_{i_1} w_{i_2} \dots w_{i_m} = x_1 x_{i_1} x_{i_2} \dots x_{i_m}$$

Modified Post

Correspondence Problem (MPCP)

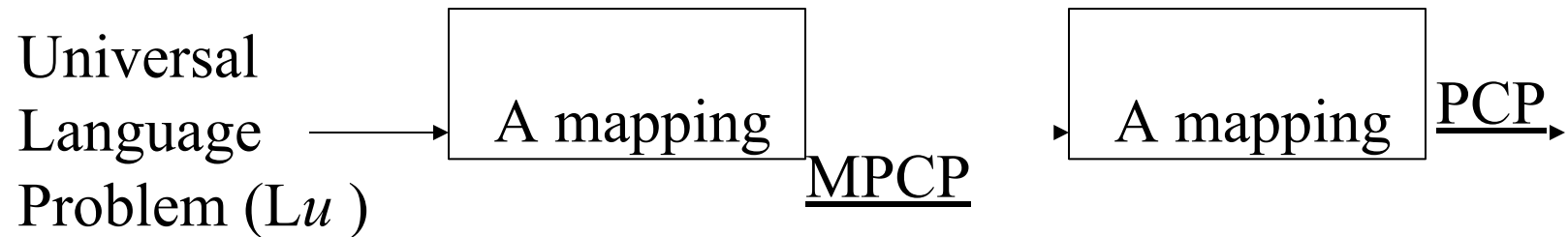
	List A	List B
i	w_i	x_i
1	10	10
2	110	11
3	11	011

**This MPCP instance has a
solution: 1,2,3**

$$w_1 w_2 w_3 = x_1 x_2 x_3$$
$$10 \ 110 \ 11 = 10 \ 11 \ 011$$

Undecidability of PCP

To show that PCP is undecidable, we will reduce the universal language problem (Lu) to MPCP and then to PCP:



If PCP can be solved, Lu can also be solved.

Lu is undecidable, so PCP must also be undecidable.

Reducing MPCP to PCP

- This can be done by inserting a special symbol (*) to the strings in list A and B of to make sure that the first pair will always go first in any solution.
- List A : * follows the symbols of Σ
- List B : * precedes the symbols of Σ
- $w_{k+1} = \$; x_{k+1} = *\$$

Mapping MPCP to PCP

Suppose the original MPCP instance is:

	A	B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

Mapping MPCP to PCP

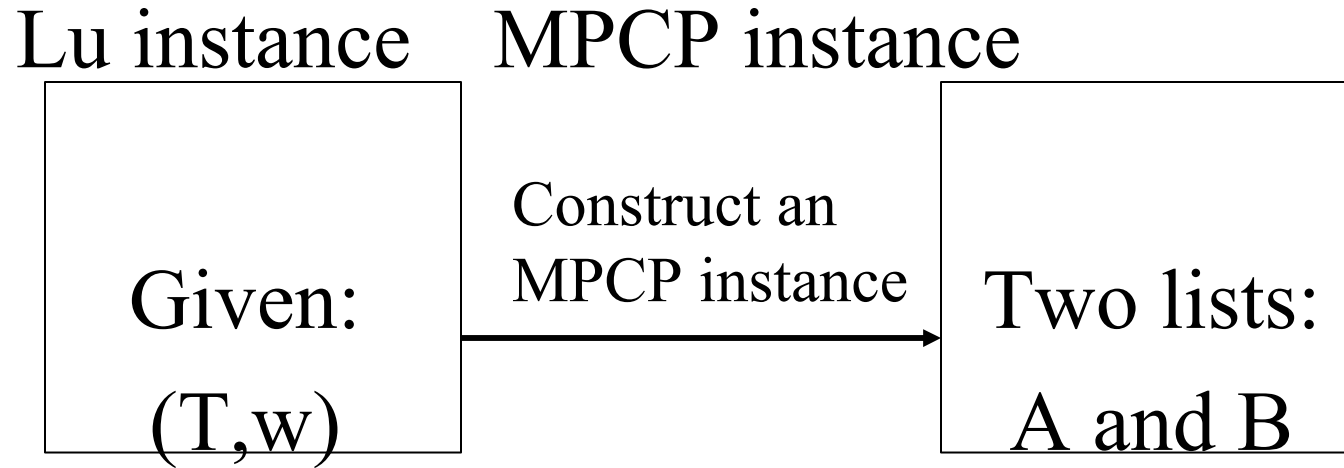
The mapped PCP instance will be:

	A	B
i	w_i	x_i
0	*1*	*1*1*1
1	1*	*1*1*1
2	1*0*1*1*1*	*1*0
3	1*0*	*0
4	\$	*\$

Mapping Lu to MPCP

- Turing machine M and an input w , we want to determine if M will accept w .
- the mapped MPCP instance should have a solution if and only if M accepts w .

Mapping Lu to MPCP



If T accepts w , the two lists can be matched. Otherwise, the two lists cannot be matched.

Rules of Reducing LU to MPCP

- We summarize the mapping as follows.
Given T and w , there are five types of strings in list A and B :
- Starting string (first pair):

List A List B

$\# \#q_0w\#$

where q_0 is the starting state of T .

- Strings for copying:

List

A X

List

B

X

where $X\#$ is any tape symbol (including the blank).

$\#$ is a separator can be appended to both the lists

- Strings from the transition function

δ : List A List B

$qX \quad Yp \quad \text{from } \delta(q,X)=(p,Y,R)$

$ZqX \quad pZY \quad \text{from } \delta(q,X)=(p,Y,L)$

$q\# \quad Yp\# \quad \text{from } \delta(q,\#)=(p,Y,R)$

$Zq\# \quad pZY\# \quad \text{from } \delta(q,\#)=(p,Y,L)$

where Z is any tape symbol except the blank.

- Strings for consuming the tape symbols at the end:

List A	List B
Xq	q
qY	q
Xq	q

where q is an accepting state, and each X and Y is any tape symbol except the blank.

- Ending string:

$$\begin{array}{ccc}
 & & \text{List} \\
 & \text{List} & \\
 & & \text{B} \quad \#
 \end{array}$$

where q is an accepting state.
 $q\#\#$

- Using this mapping, we can show that the original Lu instance has a solution if and only if the mapped MPCP instance has a solution.

PCP is undecidable

- Theorem: Post's Correspondence Problem is undecidable.
- We have seen the reduction of MPCP to PCP
- now we see how to reduce Lu to MPCP.
 - M accepts w if and only if the constructed MPCP instance has a solution.
 - As Lu is undecidable, MPCP is also undecidable.

Recursive and Recursively Enumerable Language - Properties

Recursive Enumerable (RE) or Type -0 Language

- RE languages or type-0 languages are generated by type-0 grammars.
- An RE language can be accepted or recognized by Turing machine which means it will enter into final state for the strings of language
 - May or may not enter into rejecting state for the strings which are not part of the language.
- It means TM can loop forever for the strings which are not a part of the language.
- RE languages are also called as Turing recognizable languages.

Recursive Enumerable (RE) or Type -0 Language

- RE languages or type-0 languages are generated by type-0 grammars.
- An RE language can be accepted or recognized by Turing machine which means it will enter into final state for the strings of language
 - May or may not enter into rejecting state for the strings which are not part of the language.
- It means TM can loop forever for the strings which are not a part of the language.
- RE languages are also called as Turing recognizable languages.

Recursive Language (REC)

- A recursive language (subset of RE)
- Decided by Turing machine
- It will enter into final state for the strings of language and rejecting state for the strings which are not part of the language.
 - e.g.; $L = \{a^n b^n c^n | n \geq 1\}$ is recursive because we can construct a turing machine which will move to final state if the string is of the form $a^n b^n c^n$
 - else move to non-final state.
 - So the TM will always halt in this case.
- REC languages are also called as Turing decidable languages.

Closure Properties of Recursive Languages

Union: If L_1 and L_2 are two recursive languages, their union $L_1 \cup L_2$ will also be recursive because if TM halts for L_1 and halts for L_2 , it will also halt for $L_1 \cup L_2$.

Concatenation: If L_1 and L_2 are two recursive languages, their concatenation $L_1.L_2$ will also be recursive. For Example:

$$L_1 = \{a^n b^n c^n \mid n \geq 0\}$$

$$L_2 = \{d^m e^m f^m \mid m \geq 0\}$$

$$L_3 = L_1.L_2$$

$$= \{a^n b^n c^n d^m e^m f^m \mid m \geq 0 \text{ and } n \geq 0\} \text{ is also recursive.}$$

- **Kleene Closure:** If L_1 is recursive, its kleene closure L_1^* will also be recursive. For Example:

$$L_1 = \{a^n b^n c^n \mid n \geq 0\}$$

$$L_1^* = \{a^n b^n c^n \mid n \geq 0\}^* \text{ is also recursive.}$$

- **Intersection and complement:** If L_1 and L_2 are two recursive languages, their intersection $L_1 \cap L_2$ will also be recursive. For Example:

$$L_1 = \{a^n b^n c^n d^m \mid n \geq 0 \text{ and } m \geq 0\}$$


$$L_2 = \{a^n b^n c^n d^n \mid n \geq 0 \text{ and } m \geq 0\}$$

$$L_3 = L_1 \cap L_2$$


$$= \{a^n b^n c^n d^n \mid n \geq 0\} \text{ will be recursive.}$$

Introduction to Computational Complexity

- Computational
computer

 Problems that can be
modelled and solved by

- Complexity
resource (time, space) a
problem takes up when
solved

 how much of some
being

- The amount of resources required for executing a particular (computation or) algorithm is the computational complexity of that algorithm.
- In general, when we talk about complexity we are talking about time complexity and space complexity.

Turing Machines - the Role

- Basic tool for complexity theory.
- An abstract model of computation machine: conceptually simple .
- can execute any computation possible on “known computers”
- resources consumption models well “real” computers

TIME COMPLEXITY-Turing Machine

- How long computation takes to execute
- For a Turing machine, the **time complexity** refers to the measure of the number of times the tape moves when the machine is initialized for some input symbols.
- i.e Number of machine cycles

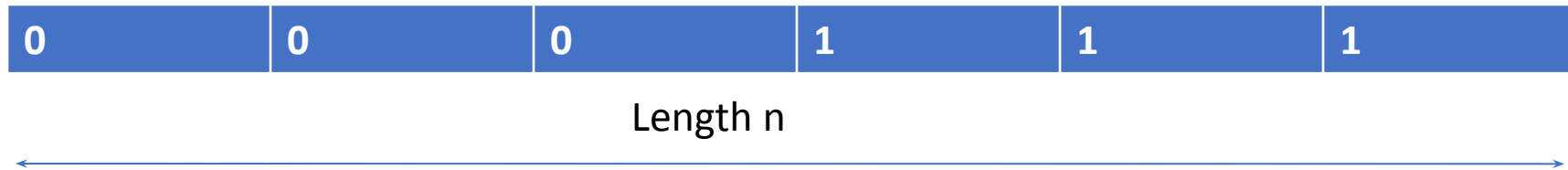
- Time complexity of Turing Machine M on an input word w , denoted as $T(M,w)$ is the number of steps done by the machine before it halts. If it does not, we set $T(M,w) = \infty$
- We can also define time complexity of a machine itself as follows:
- Function $f : \mathbb{N} \rightarrow \mathbb{N}$ is time complexity of M iff $\forall n \in \mathbb{N} :$
- $f(n) = \max\{T(M,w) : w \in \Sigma^n\}$ (assuming it halts)

SPACE COMPLEXITY-Turing Machine

- How much storage is required for computation.
- In Turing machine It is number of cells used.
- No. of bytes used

- Space used by TM M on input word w , denoted as $S(M,w)$ is defined as the number of tape cells that were visited by the head before M halted.
- If M does not halt, it is not defined. We say that $f : \mathbb{N} \rightarrow \mathbb{N}$ is space complexity of M (with stop property) iff $\forall n \in \mathbb{N} : f(n) = \max\{S(M,w) : w \in \Sigma^n\}$

EXAMPLE

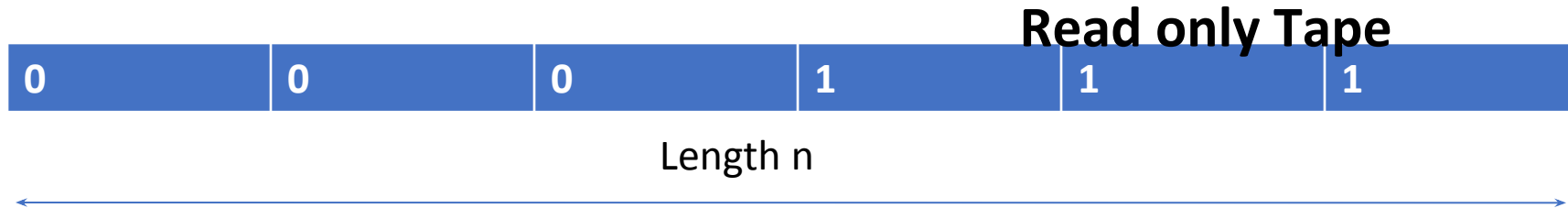


Time complexity

$\in O^n 1^n$

- Each scan takes n Steps
- The overall running time is $O(n^2)$
- Optimal solution for one tape Turing Machine $O(n \log n)$

EXAMPLE



Space complexity

$$x \in 0^n 1^n$$

- Make the input tape as read only tape
- How much space it require for computation?
- If it uses tape as a stack it requires $O(n)$ cells
- Add Binary counter logic to the TM so If $n=64$, it takes $\log_2(n)=7$ bits in binary to keep the count
- So we can solve it using $O(\log n)$ space.

NP HARD AND NP COMPLETE



P and NP problems

- P: P problems refer to problems where an algorithm would take a polynomial amount of time to solve, or where Big-O is a polynomial (i.e. $O(1)$, $O(n)$, $O(n^2)$, etc).
- These are problems that would be considered 'easy' to solve, and thus do not generally have immense run times.
- Examples:
 - a. Linear Search
 - b. Binary Search
 - c. Merge Sort
 - d. Quick sort etc.,

P and NP problems

- Can you guess the approximate time complexity of the given problem.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

NP-Hard and Complete-Need

- Np Hard and complete problems gives a base for a research topics.
- Most of the algorithms are categorized into polynomial time (linear time), exponential time (2^n) algorithms.
- Reason for looking other algorithms when we already have an algorithm
- Reducing the run time is the major challenge!
- Improve Efficiency until we have constant time Algorithm ie. $O(1)$



Polynomial Vs Exponential Algorithms

Polynomial Algorithms

Linear Search - n

Binary Search - $\log n$

Insertion sort - n^2

Merge Sort - $n \log n$

Exponential Algorithms

0/1 Knapsack Problem - 2^n

Travelling Salesman Problem - 2^n

Sum of Subsets - 2^n

Hamiltonian cycle - 2^n

Exponential Algorithms are much more time consuming than Polynomial Algorithms. The framework for converting Exponential to Polynomial Algorithms is called NP Hard and NP Complete problems.



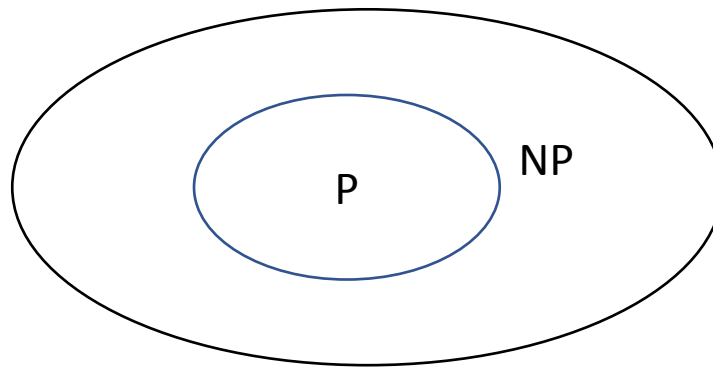
P and NP problems

- Non deterministic Polynomial time solving.
- Problem which can't be solved in polynomial time like TSP(travelling salesman problem)
- An easy example of this is subset sum: given a set of numbers, does there exist a subset whose sum is zero?.
- but NP problems are checkable in polynomial time means that given a solution of a problem , we can check that whether the solution is correct or not in polynomial time.

NP and P Problems

- P -Deterministic polynomial time Algorithms
- NP-Non Deterministic Polynomial Algorithms

For eg.Merge Sort from Non deterministic to Deterministic Algorithm



NP Hard and NP Complete

- Before knowing about NP-Hard and NP-Complete we need to know about reducibility:
- Reduction:
- Consider there are two problems, A and B, and we know that the problem A is NP class problem and problem B is a P class problem.
- If problem A can be reduced, or converted to problem B, and this reduction takes a polynomial amount of time, then we can say that A is also a P class problem (A is reducible to B).

NP Hard and NP Complete

- A problem is classified as NP-Hard when an algorithm for solving it can be translated to solve *any* NP problem.
- Then we can say, this problem is *at least* as hard as any NP problem, but it could be much harder or more complex.
- NP-Complete problems are problems that live in both the NP and NP-Hard classes.
- This means that NP-Complete problems can be verified in polynomial time and that any NP problem can be reduced to this problem in polynomial time.

How to convert Exponential Algorithms to Polynomial Algorithms?

This can be done in two ways

- By finding Deterministic and Non Deterministic Polynomial Algorithms
- Deterministic –finite solution for every run of the program
- Non Deterministic-random solution for every run of the program(every state has n number of possible next states)

Or

- By solving a base problem(which includes all Exponential problems)by finding out the relationship among them



Non Deterministic Algorithm(Magic into procedure in future)

Problem Statement : Search an element x on $A[1:n]$ where $n \geq 1$, on successful search return j if $a[j]$ is equals to x otherwise return 0.

- **Non-deterministic Algorithm for this problem :**

1. $j = \text{choice}(a, n)$ ----- $O(1)$

2. if($A[j] == x$) then

{

 write(j);

 success();----- $O(1)$

}

3. write(0);

failure();----- $O(1)$

Non Deterministic Methods

choice(X) : chooses any value randomly from the set X .

failure() : denotes the unsuccessful solution.

success() : Solution is successful and current thread terminates.



Satisfiability Problem(SAT)

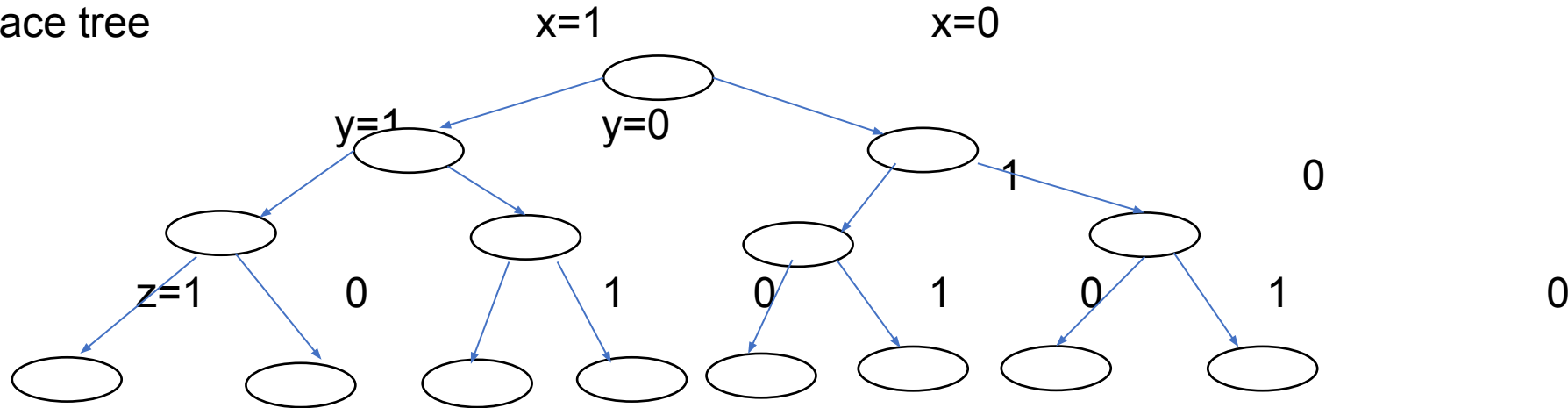
- **Definition.** A formula A is *satisfiable* if there exists an assignment of values to its variables that makes A true.

$A = (\neg x \vee y) \wedge (\sim y \vee z) \wedge (x \vee \sim z \vee y)$ - Three variables hence we have 2^3 input so this is similar to other exponential Algorithms

It take values from 000 to 111 (different combination of binary input for each variable x, y, z)

We have to try 8 values ie. 2^3 . For n variables it will be 2^n

State space tree



If this problem is solved all other exponential algorithms can be solved in polynomial time!



Np Hard-Reduction

- SAT is a Np-Hard Problem
- $SAT \longleftrightarrow 0/1 \text{ Knapsack Problem (Reduction!)}$
- Then 0/1 knapsack is also a NP Hard Problem if the reduction is in polynomial time
- $SAT \longleftrightarrow L1$, then L1 is also Np-Hard
- $SAT \longleftrightarrow L1, L1 \longleftrightarrow L2$ then L2 is also NP Hard hence transitive property holds true.

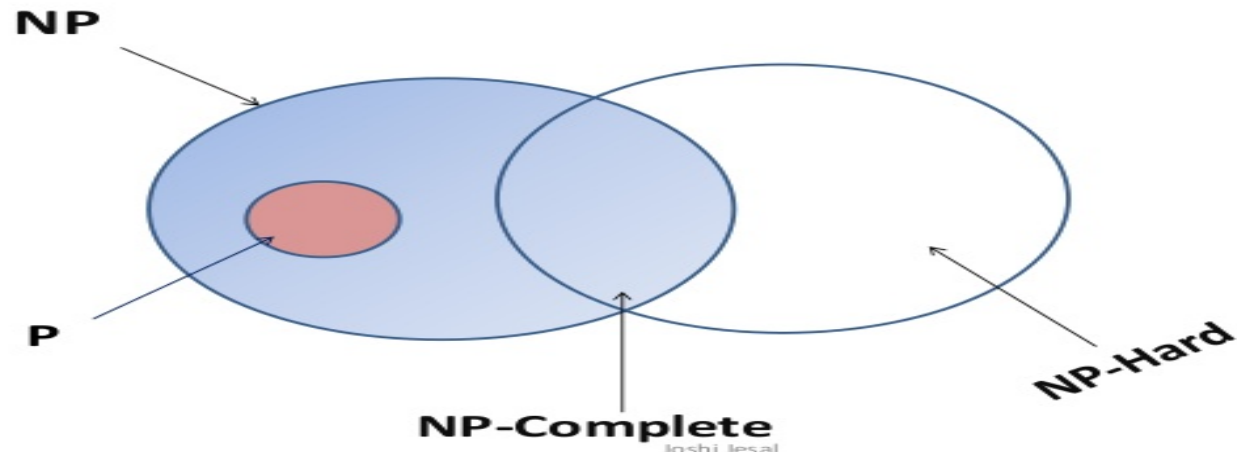


Np Complete

If a problem comes under Np Hard and if it has a non deterministic algorithm then it is also called as a Np Complete problem

SAT \longleftrightarrow L $\xrightarrow{\text{Non Deterministic}}$ NP Complete Problem

Relationship among P, NP, NP-Complete and NP-Hard



Cook's Theorem

- Cook's Theorem states that
- Any NP problem can be converted to SAT in polynomial time
- SAT will come into P if and only if $P=NP$, it will become deterministic.

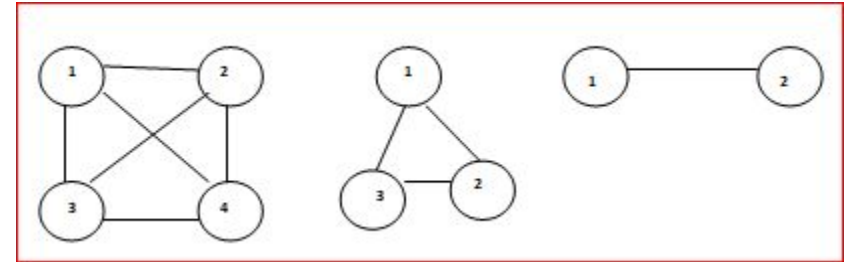
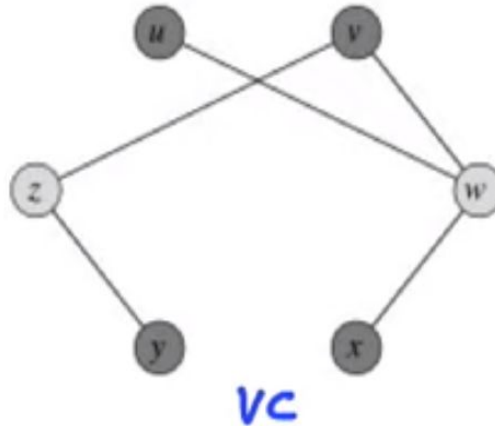
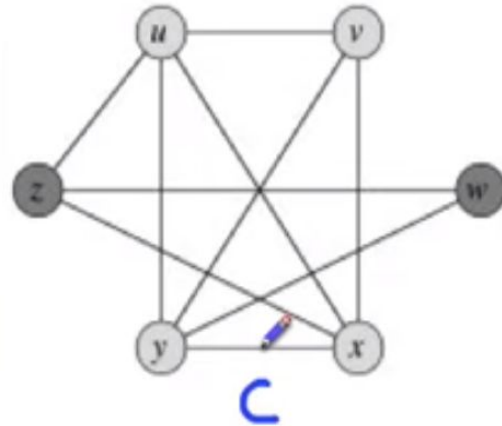


Clique to Vertex Cover

Complete Problems

Vertex Cover is NP Complete

VERTEX-COVER = $\{\langle G, k \rangle : \text{graph } G \text{ has a vertex cover of size } k\}$



There is a straightforward reduction of CLIQUE to VERTEX-COVER, illustrated in the figure.

Given an instance $G=(V,E)$ of CLIQUE, one computes the complement of G , which we will call $G_c = (V, \bar{E})$, where $(u,v) \in \bar{E}$ iff $(u,v) \notin E$.

The graph G has a clique of size k iff the complement graph has a vertex cover of size $|V| - k$.



Summary

- Polynomial and Exponential Algorithms
- Framework/Guidelines for working on Exponential Algorithms
- NP-Hard
- Np-Complete
- Cook's Theorem

Hope all our Non Deterministic Algorithms will turn into Polynomial Deterministic Algorithm i.e. $P=NP$ 😊

