




# NP HARD AND NP COMPLETE PROBLEMS



**Ayesha Nagdawala  
Sarthak Tanpure  
Shrihari Sudevan  
Sumil Suthar**

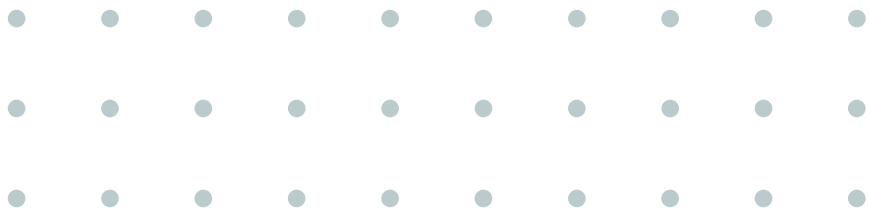
# ALGORITHMS

Polynomial time	Exponential Time
<div><div>n</div><div>-</div><div>Linear Search</div></div> <div><div>logn</div><div>-</div><div>Binary Search</div></div> <div><div>n*n</div><div>-</div><div>Insertion Sort</div></div> <div><div>n*logn</div><div>-</div><div>Merge Sort</div></div> <div><div>n*n*n</div><div>-</div><div>Matrix Multiplication</div></div>	<div><div>2^nn</div><div>-</div><div>0/1 knapsack</div></div> <div><div>2^nn</div><div>-</div><div>Travelling SP</div></div> <div><div>2^nn</div><div>-</div><div>Sum of Subsets</div></div> <div><div>2^nn</div><div>-</div><div>Graph Coloring</div></div> <div><div>2^nn</div><div>-</div><div>Hamilton Cycle</div></div>

For n=10

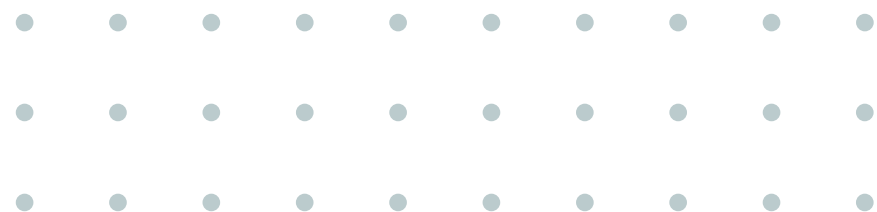
$n^2 = 10^2 = 100$

$2^n = 2^{10} = 1024$



# WHAT ARE WE TRYING TO ACHIEVE?

- **Exponential time algorithms are not considered efficient or in other words, They are too expensive to be actually used.**
- **We need to write algorithms that take polynomial time for these problems.**
- **If we can't find a solution, we need to atleast prepare a framework that will be useful for future research.**
- **Try to relate the problems and find some similarities such that if one of the problem is solved, the other can be solved as well.**
- **If writing a polynomial deterministic algorithm is not possible, try to write a Non-deterministic polynomial algorithm.**



## Deterministic algorithm:

An algorithm which we generally use like linear search, binary search where we know **how each and every step works**.

## Non-Deterministic algorithm:

Algorithm Shrihari(A,n,key)

```
{  
  i= choice();  
  if(A[i]==key)  
  {  
    print(i);  
    //success  
  }  
  print(0);  
  //failure  
}
```

—————→ O(1) assumed and may be found in the future

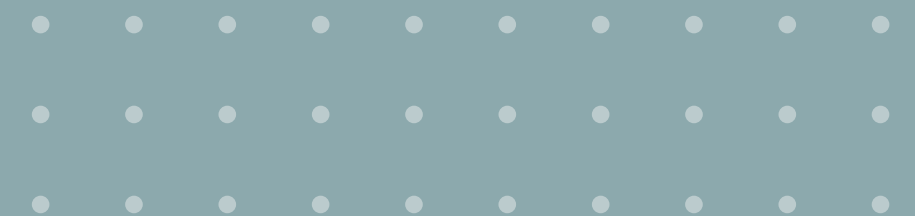
9	8	7	6	5	1
1	2	3	4	5	6

# **P-class (Polynomial time)**

A problem which can be solved in polynomial time is known as P-class problem(sorting/searching).

# **NP-class (Non-deterministic polynomial time)**

A problem which cannot be solved in polynomial time but can be verified in polynomial time is known as NP-class problem(0/1 knapsack,TSP).

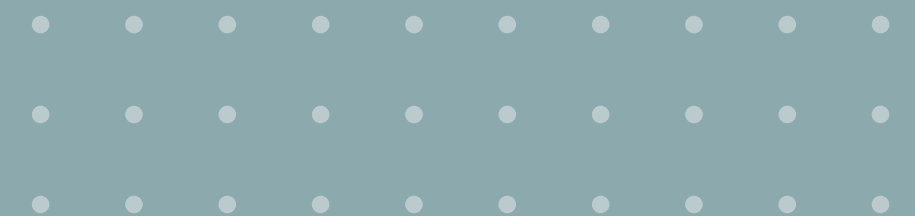


## SUDOKU

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

A 9×9 square must be filled in with numbers from 1-9 with no repeated numbers in each line, horizontally or vertically.

Also, there are 3×3 squares marked out in the grid, and each of these squares can't have any repeat numbers either.



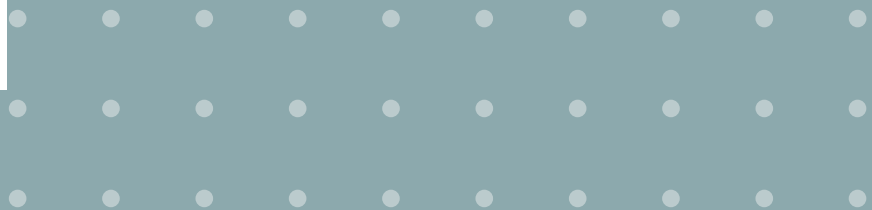
# SUDOKU3!

6  
!

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

5  
!

Solving for  
each and every  
block may take  
an exponential  
time



## SOLUTION

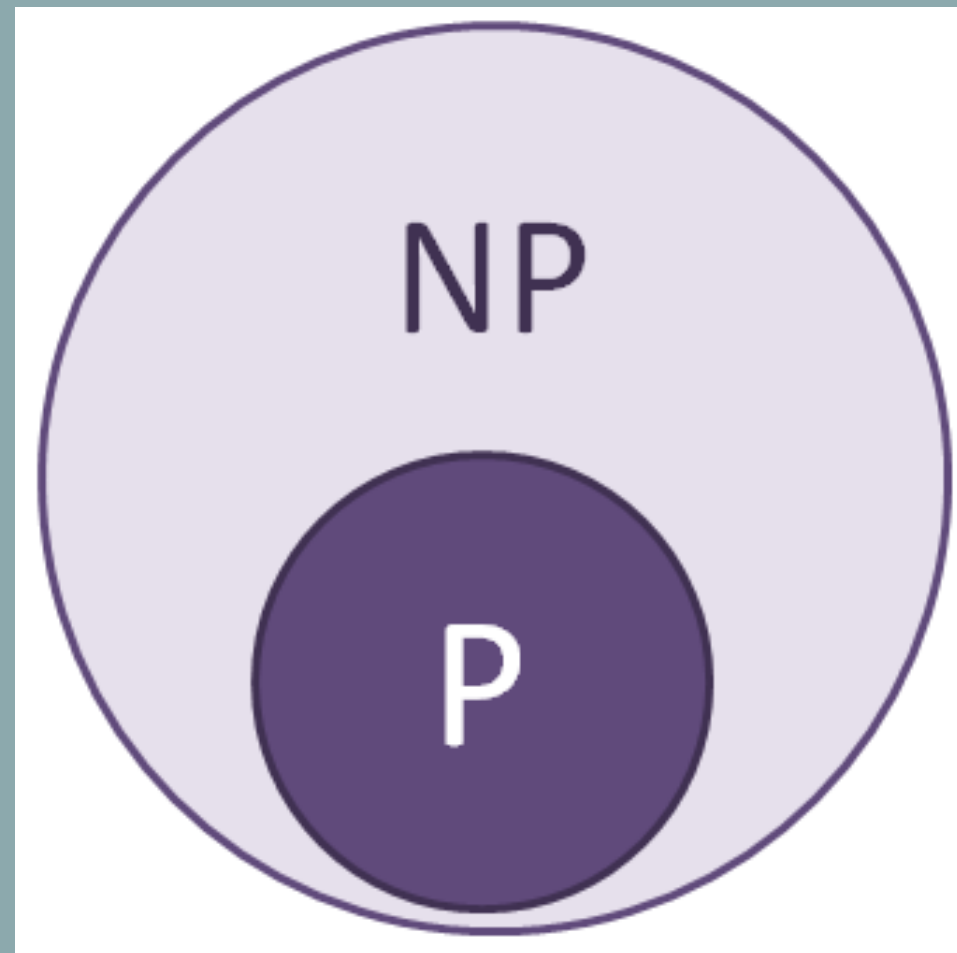
9	8	5	7	2	4	6	1	3
4	7	6	5	3	1	2	8	9
2	3	1	6	8	9	5	4	7
1	6	4	2	9	7	8	3	5
7	5	8	3	1	6	4	9	2
3	2	9	8	4	5	7	6	1
5	9	7	1	6	8	3	2	4
8	1	2	4	5	3	9	7	6
6	4	3	9	7	2	1	5	8

Not easy to solve but  
easier to verify in  
polynomial time





The deterministic algorithms we know today were at some time non-deterministic.



The non-deterministic algorithms we know today may become deterministic in the future.

So we say that,

$$\underline{P} \subseteq NP$$

# HOW TO RELATE THESE PROBLEMS TOGETHER

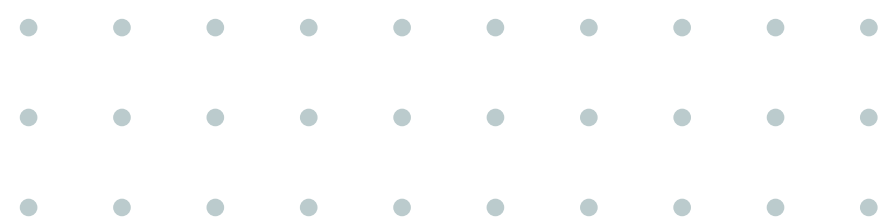
So to relate these problems we need to consider some problem as base problem!

## Satisfiability Problem(SAT):

The Satisfiability Problem is a CNF Formula.

CNF form: An expression is in CNF form(conjunctive normal form) if the set of clauses are separated by an AND ( $\wedge$ ), operator, while the literals are connected by an OR ( $\vee$ ) operator. The following is an example of an expression in the CNF form:

$$f = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_2)$$



## Exponential Time

- |       |                  |
|-------|------------------|
| $2^n$ | - 0/1 knapsack   |
| $2^n$ | - Travelling SP  |
| $2^n$ | - Sum of Subsets |
| $2^n$ | - Graph Coloring |
| $2^n$ | - Hamilton Cycle |

Given a boolean expression  $F$  having  $n$  variables  $x_1, x_2, \dots, x_n$ , and Boolean operators, is it possible to have an assignment for variables true or false such that binary expression  $F$  is true?

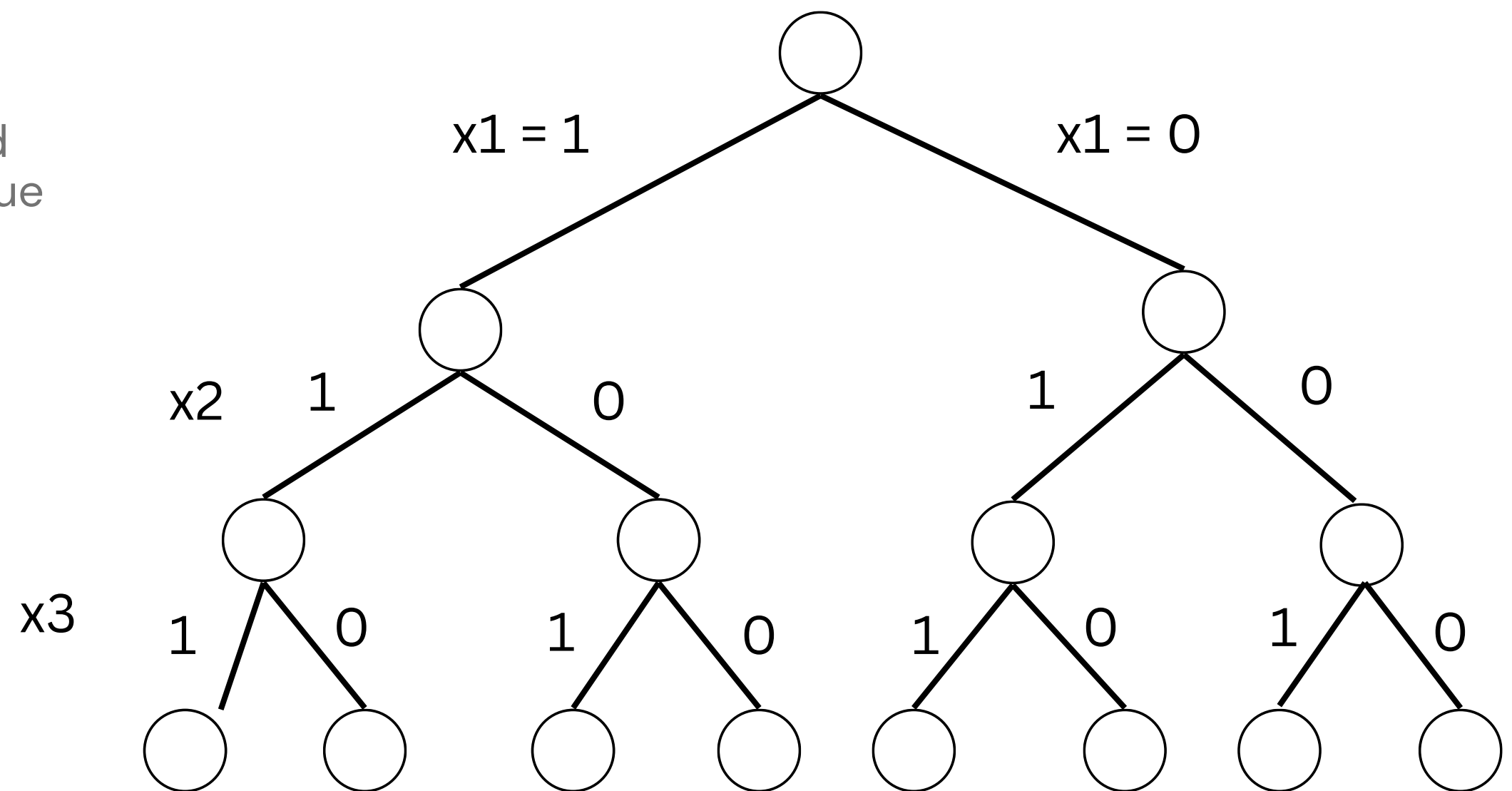
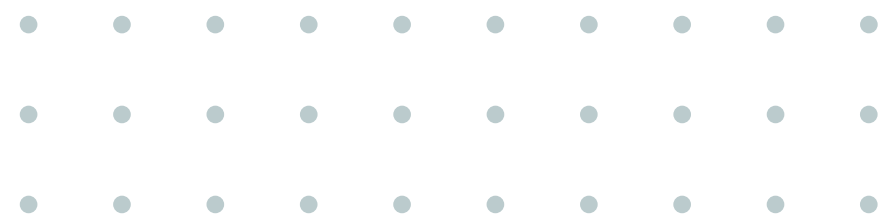
Example :

$$F = \{ (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_3 \vee x_2) \}$$

Here there are 8 possibilities and we need to check for which of these value/s is it true

X1	X2	X3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

So here we have 8 possibilities  
 $8=2^3$   
So for  $n$  variables it'll be  $2^n$   
Therefore, it is exponential time problem.



State Space Tree

# Knapsack 0/1 Problem : Maximize the profit

Example :

Profit (P) = {10,8,12}

Weight (W) = {5,4,3}

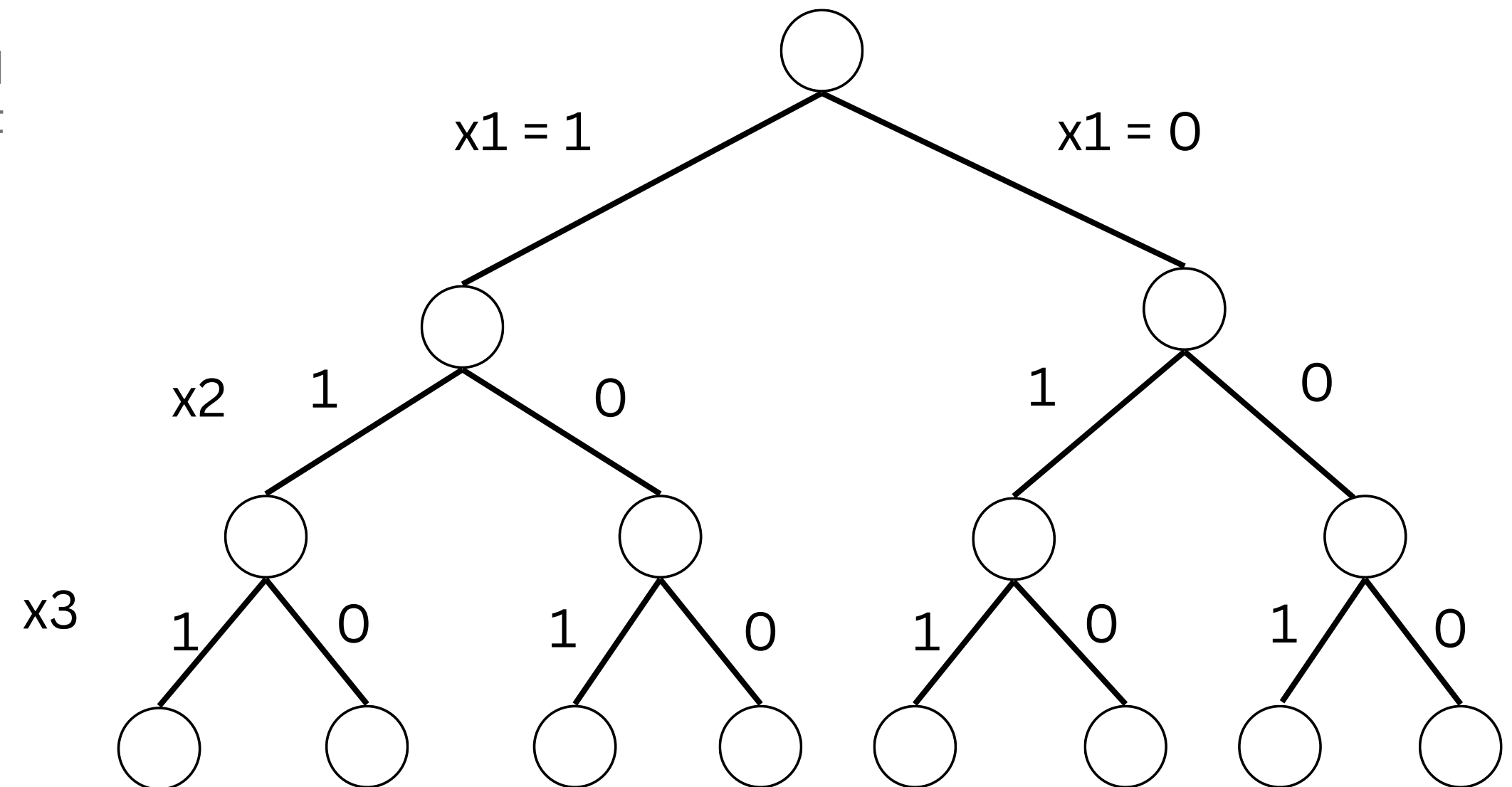
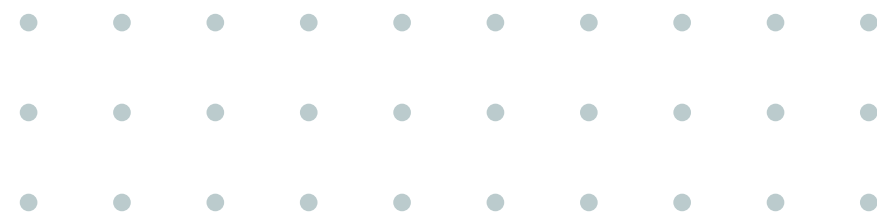
F = { 0/1, 0/1, 0/1}, n =3, m = 8

Here there are 8 possibilities and we need to check for which of these value/s we get maximum profit

X1	X2	X3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

So here we have 8 possibilities  
 $8=2^3$

So for n variables it'll be  $2^n$   
Therefore, it is exponential time problem.



State Space Tree

Now we need to show these problems are similar to satisfiability problem such that if satisfiability is solved in polynomial time all can be solved in polynomial time

This can be done using Reduction

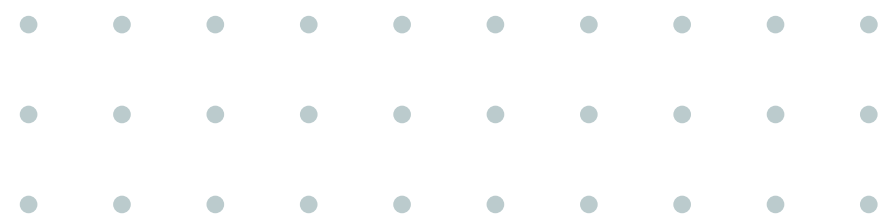
Reduction from Problem A to Problem B ( $A \leq B$ ):

- A reduction from Problem A to Problem B is an algorithm or transformation that allows you to convert an instance of Problem A into an instance of Problem B in polynomial time.
- If such a reduction exists, it means that if you can solve Problem B efficiently, you can also solve Problem A efficiently. In other words, Problem A is no harder than Problem B.

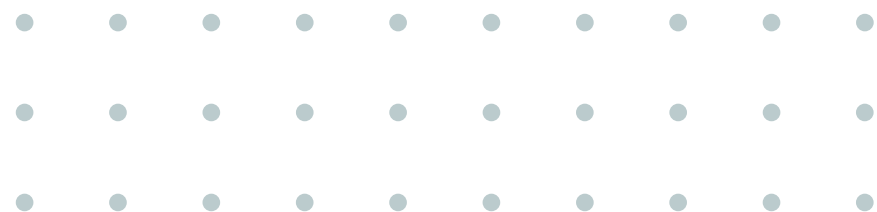
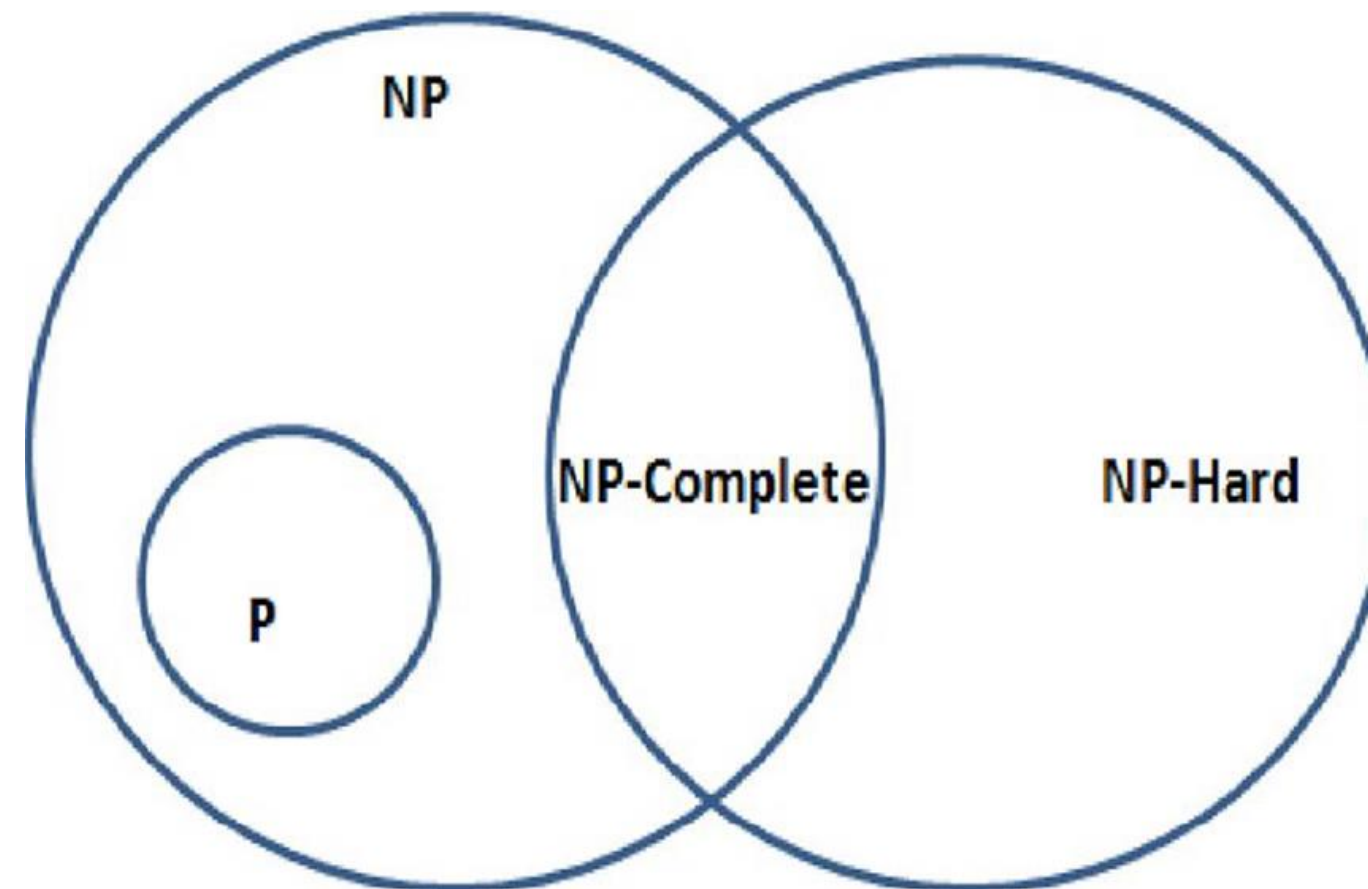
Satisfiability  $\leq$  0/1 Knapsack Problem

Reduction follows the transitivity property.

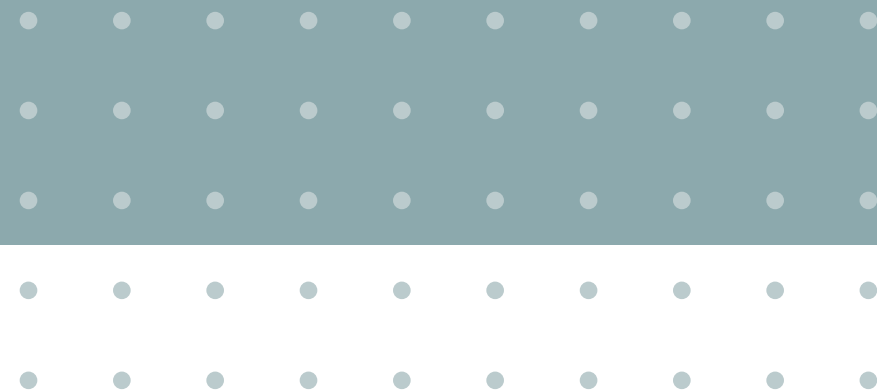
i.e if you have reductions  $A \leq B$  and  $B \leq C$ , then you can conclude that  $A \leq C$ .



- NP-hard: Class of problems are at least as hard as the hardest problems in NP. NP-hard problems do not have to be in NP; means N hard problem may not even be decidable.
- NP-complete: Class of decision problems which contains the hardest problems in N P. Each NP- complete problem has to be in NP.



# Proving that Hamiltonian cycle is NP-Complete problem





## BEFORE WE MOVE AHEAD

**P – Decision Problem**

$X - [A] - \{0, 1\}$

**NP**

$(X, \{0, 1\}, \text{Certificate})$

**A Hard  $\Rightarrow$  B Hard**

$X - [A] - \{0, 1\}$

$Y - [B] - \{0, 1\}$

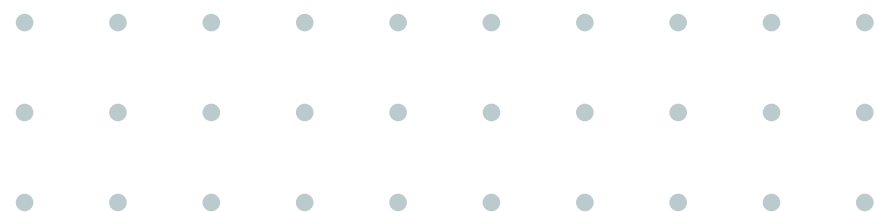
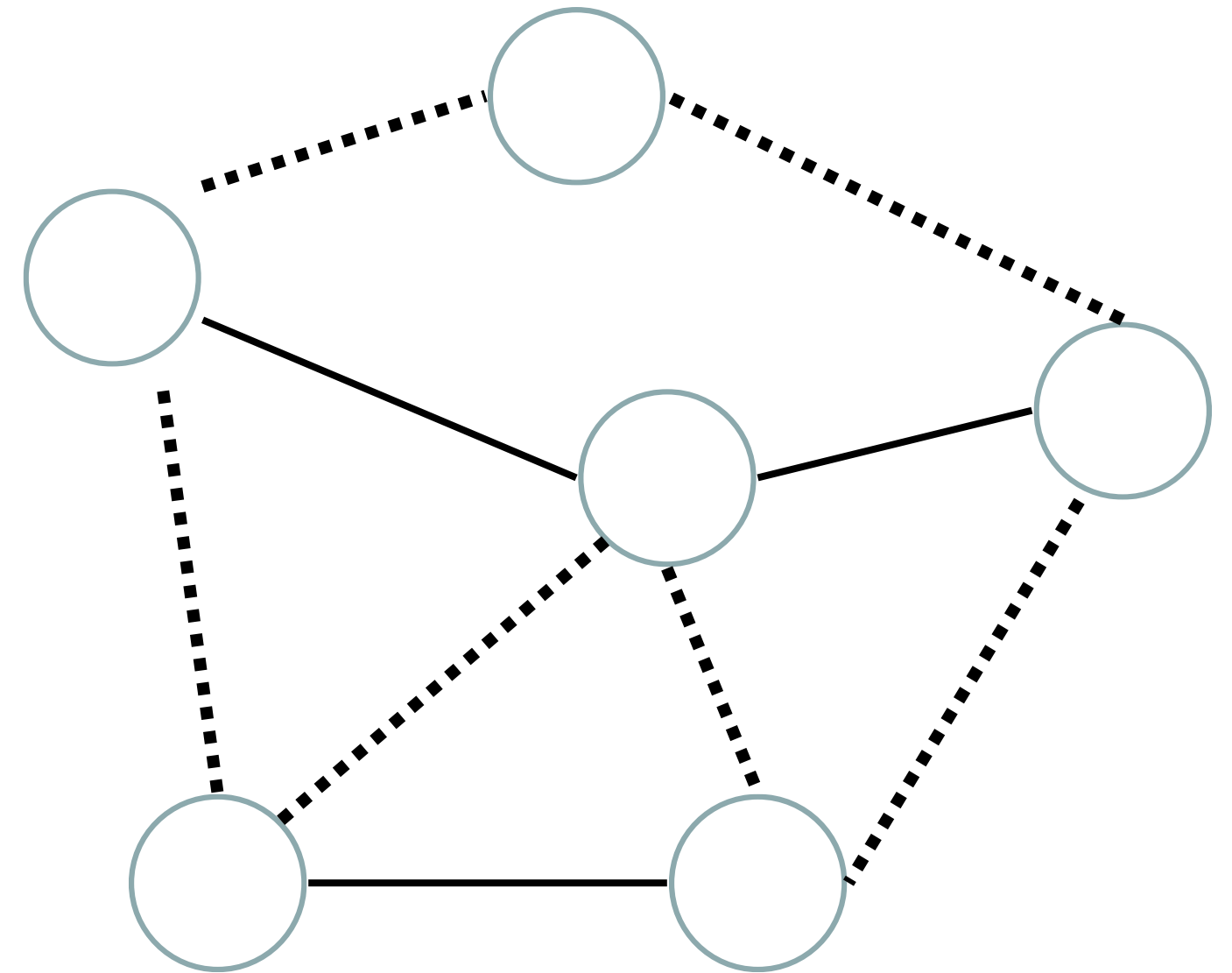
**B Easy  $\Rightarrow$  A Easy**

$R \ x \rightarrow y \quad A(x) = B(R(x))$



## A – Hamiltonian Cycle

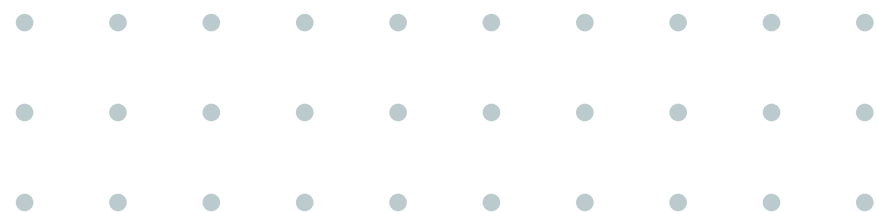
A Hamiltonian cycle is a concept in graph theory, and it refers to a specific type of cycle within a graph. More precisely, it is a simple cycle that visits every vertex in the graph exactly once and returns to the starting vertex.



# Hamiltonian Cycle is NP

- we can efficiently verify a "certificate," which is a sequence of vertices forming a Hamiltonian Cycle, given an instance of the problem (a graph  $G$  and a positive integer  $k$ ).
- Verification can be performed in polynomial time, specifically  $O(V + E)$ , where  $V$  is the number of vertices and  $E$  is the number of edges in the graph.
- The verification process checks that all vertices in the certificate are part of the graph, and that every pair of adjacent vertices in the certificate is connected by an edge in the graph.

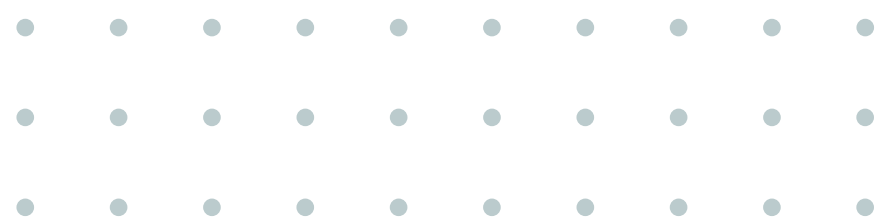
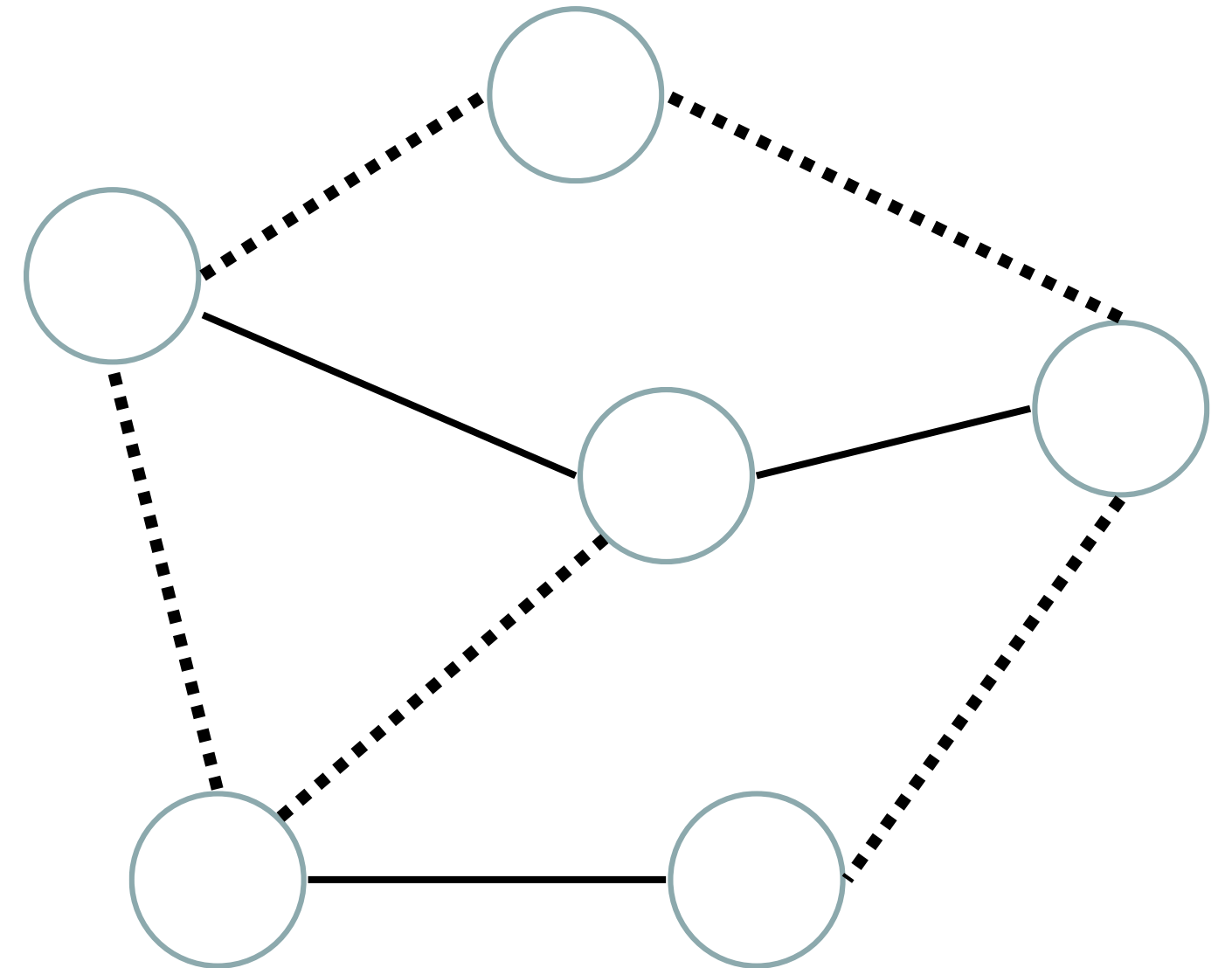
```
flag=true
For every pair {u, v} in the subset V':
    Check that these two have an edge
    between them
    If there is no edge, set flag to false
    and break
If flag is true:
    Solution is correct
Else:
    Solution is incorrect
```



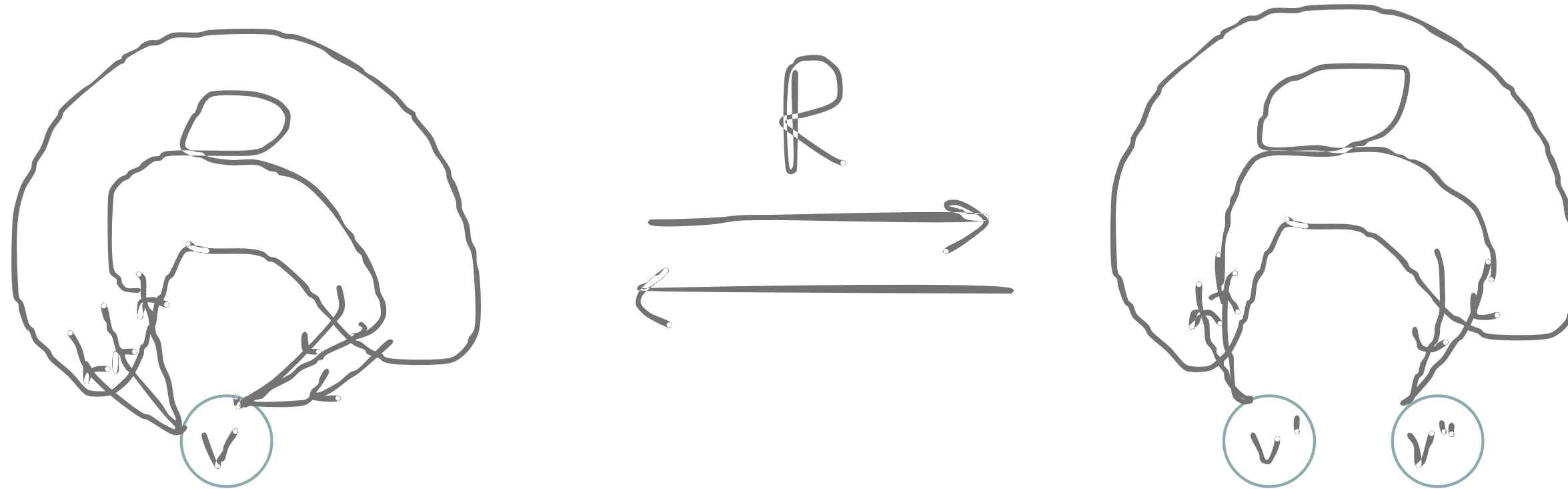
# Hamiltonian Path

A Hamiltonian path is a concept in graph theory, and it refers to a specific type of path within a graph. More precisely, it is a simple path that visits every vertex in the graph exactly once, without repeating any vertex, and without necessarily returning to the starting vertex.

In order to prove the Hamiltonian Cycle is NP-Hard, we will have to reduce a known NP-Hard problem to this problem. We will carry out a reduction from the Hamiltonian Path Problem to the Hamiltonian Cycle problem.



$$G = (V, E) \quad G' = (V', E')$$



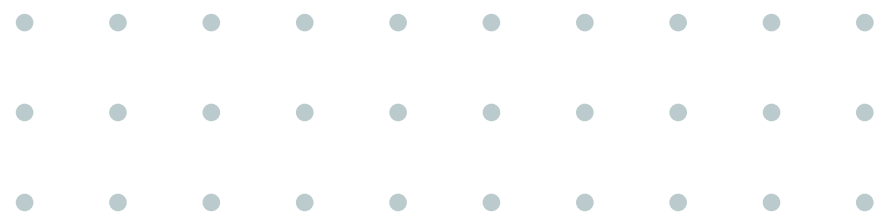
The graph  $G'$  contains a Hamiltonian Cycle if graph  $G$  contains a Hamiltonian Path. Therefore, any instance of the Hamiltonian Cycle problem can be reduced to an instance of the Hamiltonian Path problem. Thus, the Hamiltonian Cycle is NP-Hard.

Conclusion: Since, the Hamiltonian Cycle is both, a NP-Problem and NP-Hard. Therefore, it is a NP-Complete problem.

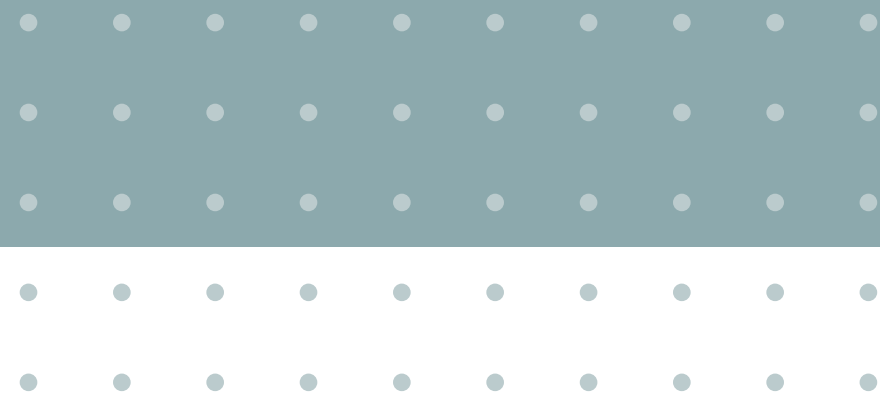


# Problem Classified as NP-Complete:

- Hamiltonian Cycle is NP-Complete
- The Traveling Salesperson Problem is NP Complete
- 0/1 Knapsack Problem is NP Complete
- Optimized Longest Path is NP Complete
- Subset Sum is NP Complete
- Set partition is NP complete
- Subset Equality is NP Complete
- Hitting Set problem is NP Complete, etc.



# Why are the Knapsack Considered NP-Complete Despite Dynamic Programming Solutions?



# 0/1 Knapsack Problem using DP

The 0/1 Knapsack problem is a classic optimization problem which can be solved using dynamic programming

Time complexity =  $O(nW)$

Where,  
'n' is the number of items and  
'W' is the max capacity

---

**Algorithm 1:** Dynamic Programming Algorithm for 0-1 Knapsack Problem

---

**Data:**  $W, v_1, v_2, v_3, \dots, v_n, w_1, w_2, w_3, \dots, w_n$

**Result:**  $M[n, W]$

$M[0, w] \leftarrow 0, \forall w$  0 to  $W$

$M[i, 0] \leftarrow 0, \forall i$  0 to  $n$

**for**  $i \leftarrow 1$  to  $n$  **do**

**for**  $w \leftarrow 1$  to  $W$  **do**

**if**  $w_i \leq w$  **then**

$M[i, w] = \max(M[i - 1, w - w_i] + v_i, M[i - 1, w])$

**else**

$M[i, w] = M[i - 1, w]$

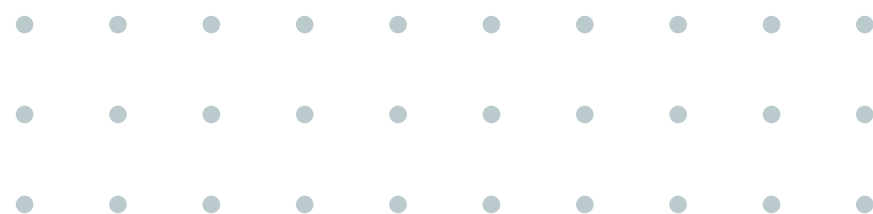
**end**

**end**

**end**

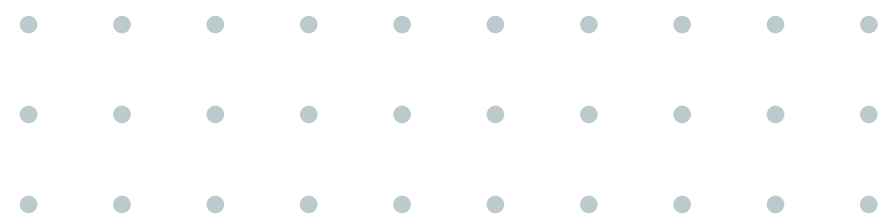
*return*  $M[n, W]$

---



# Reason why 0/1 Knapsack Problem is considered NP-Complete

- In the above decision problem, we determine if there exists a subset of items with a total weight less than or equal to 'W' and a total value greater than or equal to a given threshold.
- Therefore, we need to check all possible subsets of items to determine if there exists a subset whose total weight is less than or equal to 'W' and whose total value is greater than or equal to a given threshold.
- The number of possible subsets to check grows exponentially with the number of items, and as it is decision problem which makes it an NP-Complete problem.
- Hence, the dynamic programming solution efficiently handles the 0/1 Knapsack Problem with a fixed 'W,' its NP-Completeness becomes evident when we consider the decision version with variable 'W' and the need to find the optimal subset of items that satisfy specific criteria. This computational complexity has important implications for both theory and practical applications in various domains.





# Some Application of NP-Hard and NP-Complete:

**1. Optimization Problems:** Many real-world optimization problems are NP-hard or NP-complete. These include the traveling salesman problem (TSP), the knapsack problem, the job scheduling problem, and the vehicle routing problem. Applications include route planning for delivery trucks, resource allocation, and project scheduling.

**2. Circuit Design:** The problem of finding the optimal layout of components on an integrated circuit can be NP-hard. Designers use heuristic methods and approximations to tackle these problems efficiently.

**3. Network Design:** Designing efficient networks, such as data communication networks or transportation networks, often involves NP-hard problems. This includes problems like finding the optimal routing for data packets or planning the best transportation routes.

**4. Machine Learning:** Some machine learning problems, like feature selection and hyperparameter optimization, can be reduced to NP-hard or NP-complete problems. Researchers use approximations and heuristics to make these problems computationally tractable.





THANK YOU

