Complexity Classes

Decision Problem

- There are many problems for which the answer is a Yes or a No. These types of problems are known as **decision problems**.
- For example,
    - Finding cycle in a graph is not a decision problem, whereas checking a graph has cycle or not is a decision problem.

Decidable and Un-decidable Problem

- **Un-decidable problems** are decision problem for which it is impossible to construct single algorithm that always leads to correct 'yes' or 'no' answer for all cases of inputs (i.e) decision problem that cannot be solved by an algorithm for all cases of inputs.
- Decision problem that can be solved by an algorithm for all cases of inputs is called **Decidable Problem.**

Polynomial Time Algorithm

- For input size ***n***, if worst-case time complexity of an algorithm is $O(n^k)$, where ***k*** is a constant, the algorithm is a polynomial time algorithm.
- Polynomial time examples: O(n^2), O(n^3), O(1), O(n long n)
- Not in polynomial time examples: O(2^n), O(n!)

Computational Complexity Classes

1. **P-Class**

- The class P consists of those problems that are solvable in polynomial time, i.e. these problems can be solved in time $O(n^k)$ in worst-case, where **k** is constant.
- These problems are called **tractable**, while others are called **intractable or super polynomial**.

2. NP-Class

- NP problems are basically non-polynomial time problems. Solution can be obtained in super polynomial time.
- The class NP consists of those problems that are verifiable in polynomial time i.e. problems whose witness or certificate can be verified in polynomial time.
- This means that for the given instance of problem and a certificate to the answer being 'yes', we can check that it is correct in polynomial time.
- Every problem in this class can be solved in exponential time using exhaustive search.
- Example-1:
  - S = {3, 9, 2, 10}; t = 12; S' = {3,9}
  - Does this subset S' add to t or not?
  - Yes / No type problem (Decision problem).
  - S' is the certificate. You can verify the certificate in polynomial time. But basically finding the subsets from the given set whose sum is equal to t is not a polynomial time algorithm.
- Example-2:

- Given a graph with 20 vertices, does vertices 5, 11, 12, 16, 17 create a cycle or not?
- Yes / No type problem (Decision problem).
- Vertices 5, 11, 12, 16, 17 are the certificate. You can verify the certificate in polynomial time. But basically, finding cycles in a graph is not a polynomial time algorithm.

3. NP-Complete and NP-Hard Class

- A Problem L is **NP-complete** if
  1. $L \in NP$,
  2. $L' \leq_P L$ for every $L' \in NP$.
- L' is the known NP problem. $L' \leq_P L$ means L' should reduce to L which is also NP.
- Reduction Example: computing the median reduces to sorting.
  - If you know an algorithm for sorting, can you use that to compute median?
  - Answer is 'Yes'.
  - Sort the array, pick the middle element and output the median.
  - If you know sorting, you can compute median.
  - Computing median can be reduced to sorting.
  - Reduction means converting one problem into another problem.
- If a problem L satisfies property 2, but not necessarily property 1, we say that L is **NP-hard**.
- For Np-hard problems, polynomial time verification not possible. But reduction in polynomial time is possible.

Cook's Theorem

Cook's Theorem states that "Any NP problem can be converted to SAT problem in polynomial time".

Boolean Satisfiability Problem (SAT)

Boolean Satisfiability or simply **SAT** is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.

- **Satisfiable :** If the Boolean variables can be assigned values such that the formula turns out to be TRUE, then we say that the formula is satisfiable.
- **Unsatisfiable :** If it is not possible to assign such values, then we say that the formula is unsatisfiable.
  - **Examples:**
    - F = A ∧ ⌝B, is satisfiable, because A = TRUE and B = FALSE makes F = TRUE.
    - G = A ∧ ⌝A, is unsatisfiable, because

A	⌝A	G
T	F	F
F	T	F

Conjunctive Normal Form (CNF)

- Boolean formula is in **conjunctive normal form** (**CNF**) if it is a conjunction of one or more clauses where a clause is a disjunction of literals. Simply we say, **AND of ORs**.
- Example:

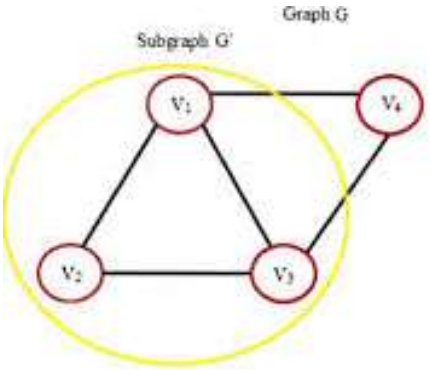
  (x1 ∨ x2 ∨ ⌝x3) ∧ (⌝x1 ∨ ⌝x2) ∧ (x1 ∨ ⌝x2) ∧ (x1 ∨ x2)

3-CNF-SAT (or) 3-SAT

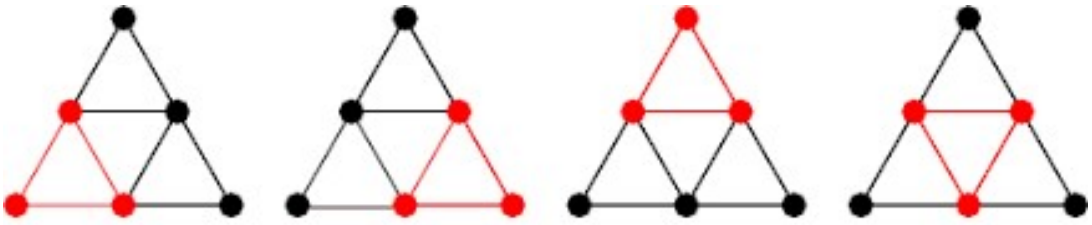
- A boolean formula is in **3**-conjunctive normal form, or **3-CNF-SAT**, or **3-SAT** if each clause has exactly three distinct literals.
- For **example**, the boolean formula (x1 ∨ x2 ∨ ¬x3) ∧ (¬x1 ∨ ¬x2 ∨ x3) ∧ (x1 ∨ ¬x2 ∨ x3) is in **3-CNF-SAT**.

Clique

- Clique problem is the computational problem of finding cliques.
- Complete subgraph of a graph is called clique.
- Complete subgraph - subsets of vertices, all adjacent to each other.
- Example -1: Subgraph G' is a complete subgraph. So we can call it as clique.



- Example-2

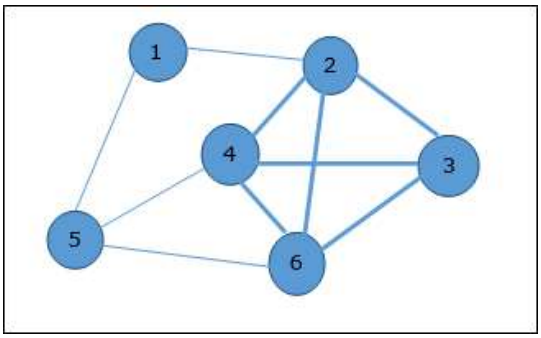


cliques

- Size of clique will be the number of vertices it contains.
- Example-1: Size of clique = 3; Example-2: Size of all cliques = 3

Maximum Clique Problem

- The Max-Clique problem is the computational problem of finding maximum clique of the graph.
- Given a small graph with **N** nodes and **E** edges, the task is to find the maximum **clique** in the given graph.
- The maximal clique is the complete subgraph of a given graph which contains the maximum number of nodes.
- Example:

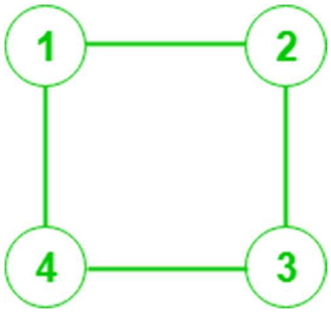


  - Here, the sub-graph containing vertices 2, 3, 4 and 6 forms a complete graph.
  - Hence, this sub-graph is a **clique**.
  - As this is the maximum complete sub-graph of the provided graph, it's a **4-Clique** (k-Clique). 'k' is the size of the clique. Here, 4 is the size of the maximum clique.

Independent Sets:

- A set of vertices I is called independent set if no two vertices in set I are adjacent to each other or in other words the set of non-adjacent vertices is called independent set.

- It is also called a **stable set.**
- The parameter $\alpha_0(G) = \max \{|I|: I \text{ is an independent set in } G\}$ is called **independence number** of G i.e the maximum number of non-adjacent vertices.



For above given graph G, Independent sets are:

$I_1 = \{1\}$, $I_2 = \{2\}$, $I_3 = \{3\}$, $I_4 = \{4\}$

$I_5 = \{1, 3\}$ and $I_6 = \{2, 4\}$

Therefore, maximum number of non-adjacent vertices i.e. Independence number $\alpha_0(G) = 2$.