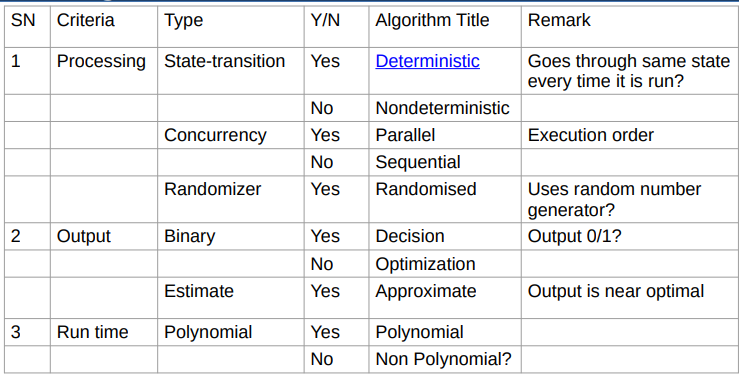
CHAPTER 6

* Algorithm classification based:
* Deterministic VS Non-Deterministic:

A **deterministic** process is one where the machine goes through the same state every time the program is run.

In a **non-deterministic**, every time the program is run, the state through which the machine goes will be different.

* Decision vs. Optimization:

**Decision** problems involve producing a binary output - typically Yes/No, True/False, or 0/1.

Example: Determining if a given number is prime (Yes/No).

**Optimization** problems involve finding the best solution based on some criterion.

Example: The Knapsack problem, where you want to maximize profit within a given weight capacity, sorting, Matrix multiplication.

Decision problems are simpler than optimization problems.

* Can we express optimization problems in terms of decision problems?

Yes

* Optimization Algorithms in terms of Decision Algorithms:

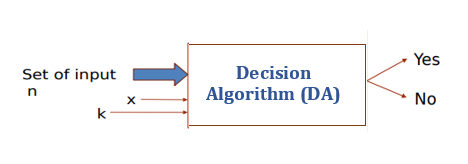
1] Sorting:

* Set of Input: n​(a set of input elements).
* Decision Algorithm (DA):

Is the given number *k*th smallest number in the set of

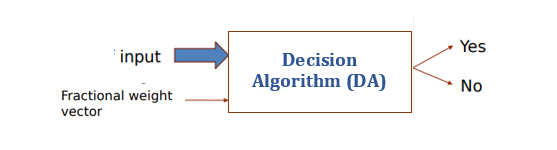
*n* numbers?

* Explanation:
  + Sorting involves repeatedly solving this decision problem for different values, determining the *k*th smallest element.
  + k=is rank
  + x=current element



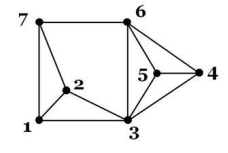
2] Knapsack

* Set of Input: *n* items with capacity, profit, and weight.
* Fractional Weight Vector: Vector representing the fractions of selected items.
* Decision Algorithm (DA):
  + Do the fractions of the selected items give a profit of 'p'?
* Explanation:
  + The decision problem is whether the selected items, considering their fractions, lead to the desired profit.



* Some Problems:

1. Clique :

* A complete sub-graph of a given graph.
* Cliques {1,2,7} ,{1,2,3}…. {3, 4, 5, 6}
* Max clique is no of elements in biggest cliques i.e. in {3, 4, 5, 6} which is 4.

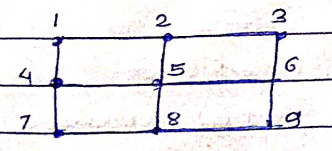
1. Circuit satisfiability (CKT-SAT):

For any given a combinational circuit (composed of AND, OR, NOT gates), is it satisfiable (produces output as 1)?

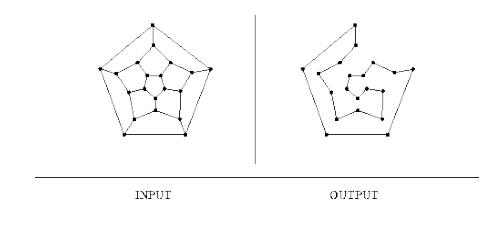
1. Satisfiability:
2. 3 Conjunctive Normal Form Satisfiability (3CNF-SAT):

* A Boolean expression in which every clause has exactly 3 literals.
* *X*=(*A*+*B*+*C*).(*A*+*B*+*D*).(*A*+*B*+*C*)

1. Vertex cover(VC):

* Subset of vertices which covers all the edges of graph.
* VC={2, 4, 6, 8}

1. Hamiltonian Cycle(HAM-CYCLE):

* Simple cycle (every vertex is visited exactly once)

1. Travelling salesman Problem (TSP):

* Builds on HCP(Hamiltonian Cycle Problem)
* We have to find smallest HCP.
* Polynomial (P) VS Non-Deterministic Polynomial(NP):

P (Polynomial Time):

1. Problems in P are those that can be solved in polynomial time.
2. The running time of the algorithm to solve the problem is polynomial in terms of the input size.

NP (Non-Deterministic Polynomial Time)

1. Problems in NP can be verified in polynomial time, they can’t be solved in polynomial time.
2. Verification of a solution is efficient.
3. While we don't know if solutions can be found efficiently, if given a solution, we can verify it in polynomial time.

Shortest vs Longest Path:

* + Shortest Path: P
  + Longest Path: NP

Euler vs Hamiltonian Cycle:

* + Eulerian Cycle: Polynomial time.
  + Hamiltonian Cycle: NP

If decision problem can't be solved in P time, then its optimization counterpart can also not solved in P time.

If decision problem can be solved in P time, then its optimization counterpart may be solved in P time.

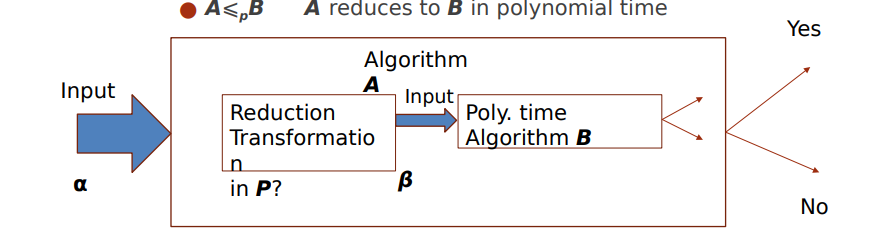
**Reduction:**

Can we solve problem A using the solution of problem B?

* A is the problem we want to solve, and B is a problem with a known solution in polynomial time (P).
* If we can solve problem A in terms of problem B, such that the solution to B helps us find the solution to A, then A ⩽p B.

Process:

* Assume we have an algorithm to solve problem A.
* Suppose problem B has a known solution, and this solution can be found in polynomial time.
* If we can somehow use the solution to problem B to efficiently solve problem A, then A ⩽p B is established.

A polynomial-time reduction implies that solving problem A is no harder than solving problem B. If we have an efficient solution for B, we can use it to efficiently solve A.

* NPC:

NP-complete (NPC) problems are among the most challenging problems in NP (non-deterministic polynomial time).

1. Start with a known NPC problem X.
2. Take an unknown problem Y.
3. Showing Y is as Hard as X.
4. Y can be mapped on X in polynomial time.
5. Other problem Z can be mapped on Y and so on…...

If X belong to NPC, Y problems also belong to NPC and Z ….belongs to NPC.

1. A problem in NPC has a property that if it can be solved in polynomial time then all other problems in NPC can also be solved in polynomial time.
2. Problems in NP can be reduced to NP-complete problems.

* NPH:

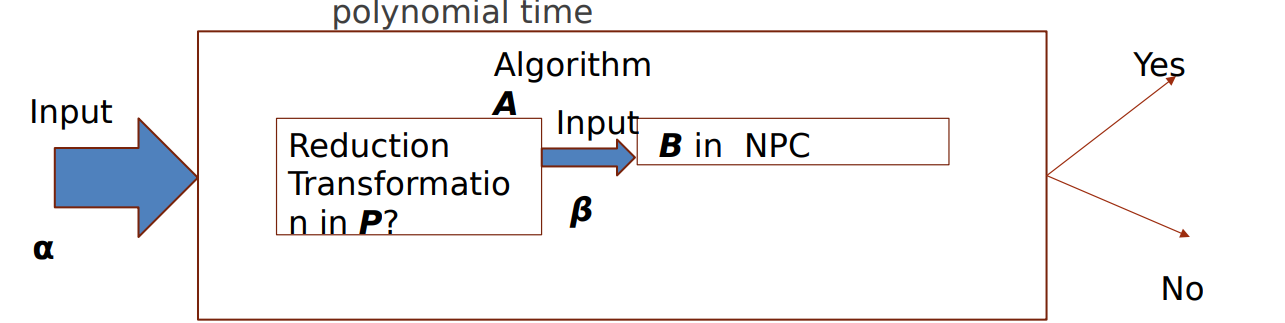
NP-hard problems are at least as hard as the hardest problems in NP.

Halting problem and subset sum.

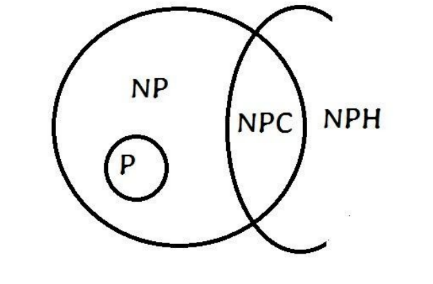
1. If a problem X is NPH, then every problem A, B, C… in NP can be reduced to X in polynomial time.
2. Can be solved in polynomial time then all NPC problem can be solved in polynomial time.

* Reductions for NP-complete problems:
* Start with a problem B that is known to be NP-complete (NPC).
* Take an unknown problem A
* A belongs to NPC if

1. We can solve A in terms of B
2. Inputs of A can be transformed into inputs of B in polynomial time.



* **Venn Diagram:**



**NP (Non-deterministic Polynomial time):** Class of decision problems for which solutions are not in p time but can be verified polynomial-time.

Ex: Hamiltonian cycles, Sudoku solutions.

**P (Polynomial time):** Decision problems solvable in polynomial time.

Ex: Finding the shortest path in a graph.

**NPC (NP-Complete):**

Subset of NP containing the hardest problems; if any NPC problem is solved in polynomial time, all problems in NP can be solved in polynomial time.

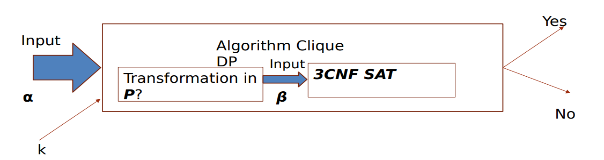
Ex: The traveling salesman problem, Boolean satisfiability problem.

**NPH (NP-Hard):**

Class of problems at least as hard as the hardest problems in NP; solving any NPH problem in polynomial time implies solving all problems in NP in polynomial time.

Ex: The Halting problem, subset sum.

1] Clique decision Problem (CDP) is in NPC:

To Prove: Does graph has clique of size k?

Given: 3CNF SAT belongs to NPC

Steps:

* + 1. **Show that CDP belongs to NP:**

Given a (certificate) set S of k nodes representing a complete sub-graph it can be verified in polynomial time. (Check that each vertex in S is connected to the other k-1 vertices) which requires O(k^2) time.

* + 1. **Show that CDP can be mapped on 3CNF SAT**

Ex:

* + - * Take a boolean equation, in 3CNF form

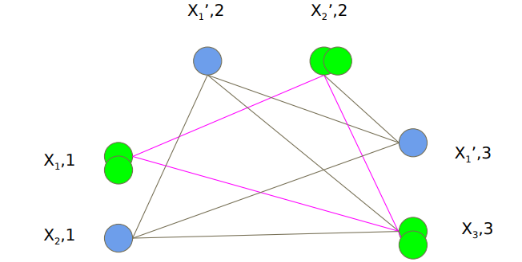
(X1+X1+X2)(X1’ + X2’ + X2’)(X1’ + X3 + X3)……. 3 variables, 3 clauses.

* + - * F= (X1^X2 ) V(X1 ’^X2 ’)V(X1 ’^X3 )

* + 1. **Construct Graph:**
  + Every literal represents a vertex
* Every clause represents a group of vertex (clause is collection of literals).
* Add edges in the graph between vertices:

- The two vertices belong to different group

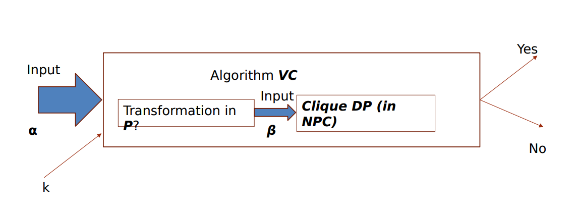
- Vertices are not complement of each other



**The reduction demonstrates that if 3CNF SAT with k clauses is satisfiable, then the corresponding graph has a clique of size k.**

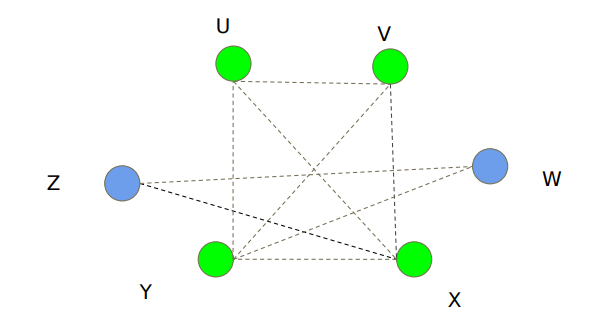
2] Vertex cover (VC) Problem is in NPC:

To Prove: Does graph has VC of size k?



Given: CDP is in NPC.

Steps:

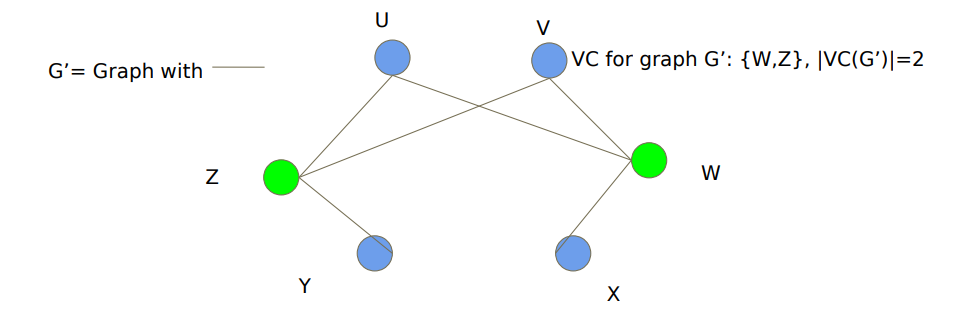
* 1. Set S consisting of k nodes in the VC.
  2. Show that VC belongs to NP.(Each edges of graph can be checked if it is incident on one of the k vertices O(|E|.k) time)
  3. Show that VC can be mapped on Clique which in NPC in polynomial time.

Let G be the given graph

Clique for Graph G: {U, V ,X ,Y}

|CLIQUE(G)| =4

* 1. Complement of G:



3] Vertex cover (VC) Problem is in NPC: