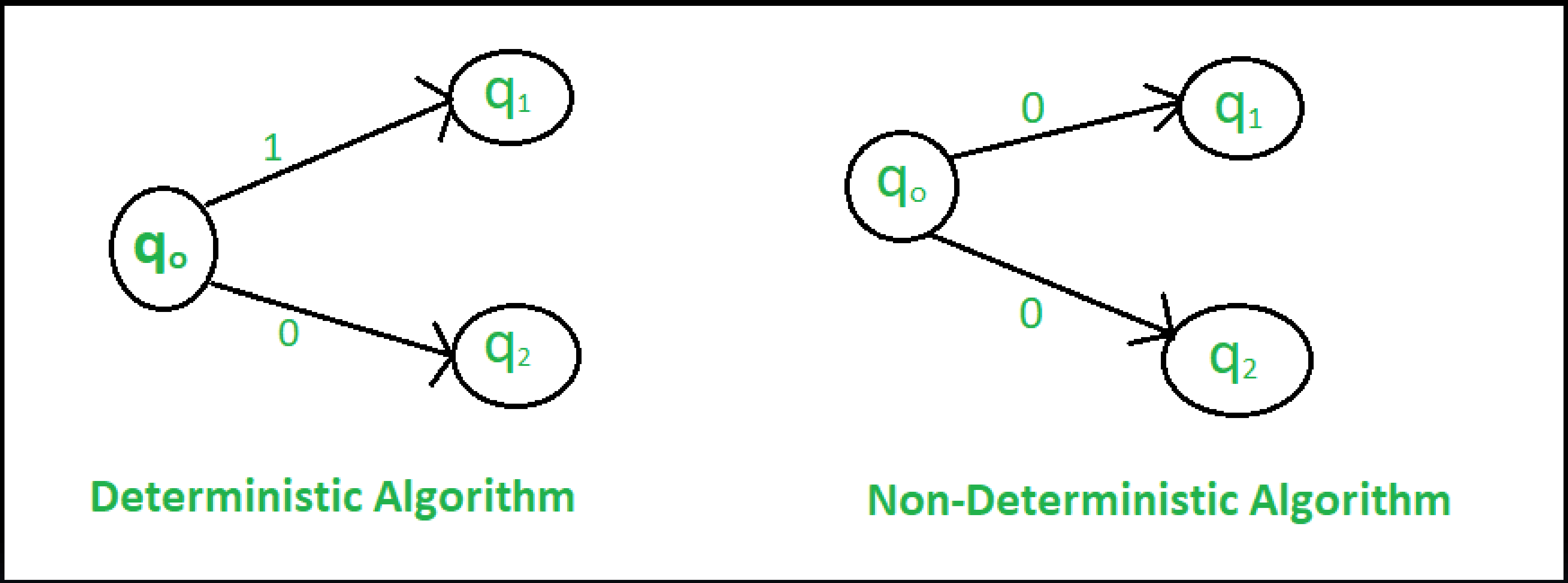
UNIT V

- Non Deterministic Algorithms
- NP Hard and NP Complete Problems

© 2022, S.Vineela Krishna, CSIT, CVR

NON DETERMINISTIC ALGORITHMS

- **DETERMINISTIC ALGORITHMS:**
  - **"Has the property that the result of every operation is uniquely defined."**
  - A problem can be solved using deterministic algorithms in **polynomial time**.
  - A deterministic algorithm is an algorithm that, given the same input and initial conditions, **will always produce the same output and follow the same sequence of steps**.
- **NON DETERMINISTIC ALGORITHMS:**
  - **"Algorithms that contain operations whose outcomes are not uniquely defined but are limited to specified sets of possibilities."**
  - For the same input, the algorithm may produce **different output** in different runs.
  - A non-deterministic algorithm **cannot solve a problem in polynomial time**


q0
q1
q2
1
0
Deterministic Algorithm
q0
q1
q2
0
0
Non-Deterministic Algorithm

- To specify such nondeterministic algorithms, we use 3 functions:
  1. **Choice(S)** - arbitrarily choose one of the elements of set S
  2. **Failure()** - signals an unsuccessful completion
  3. **Success()** - signals a successful completion

  - The assignment **X := choice(1:n)** could result in X being assigned any value from the integer range [1..n].
  - There is no rule specifying how this value is chosen.

© 2021, S.Vineela Krishna, CSIT, CVR

- The nondeterministic algorithms terminates **unsuccessfully** iff there is no set of choices which leads to the successful signal.
- The **computing time for failure and success** is taken to be **O(1)**
- **"A machine capable of executing a nondeterministic algorithms are known as nondeterministic machines"**
- "Nondeterministic machines does not make any copies of an algorithm every time a choice is to be made. Instead it has the ability to correctly choose an element from the given set".

Examples for Nondeterministic Algorithms with polynomial time

1. Searching an element x in a given set of elements A(1:n).

- We are required to determine an index 'j' such that A(j) = x or j = 0 if x is not present

```
j := Choice(1, n);
if A[j] = x then {write (j); Success();}
write (0); Failure();
```

- The time complexity is O(1)

© 2021, S.Vineela Krishna, CSIT, CVR

2. Nondeterministic 0/1 knapsack algorithm:

- the resulting profit should be at least ‘r’

```
Algorithm DKP(p, w, n, m, r, x)
{
    W := 0; P := 0;
    for i := 1 to n do
    {
        x[i] := Choice(0, 1);
        W := W + x[i] * w[i]; P := P + x[i] * p[i];
    }
    if ((W > m) or (P < r)) then Failure();
    else Success();
}
```

- The time complexity is O(n)

© 2021, S.Vineela Krishna, CSIT, CVR

3. Sort n positive integers:

```
Algorithm NSort(A, n)
// Sort n positive integers.
{
    for i := 1 to n do B[i] := 0; // Initialize B[ ].
    for i := 1 to n do
    {
        j := Choice(1, n);
        if B[j] ≠ 0 then Failure();
        B[j] := A[i];
    }
    for i := 1 to n − 1 do  // Verify order.
        if B[i] > B[i + 1] then Failure();
    write (B[1 : n]);
    Success();
}
```

© 2021, S.Vineela Krishna, CSIT, CVR

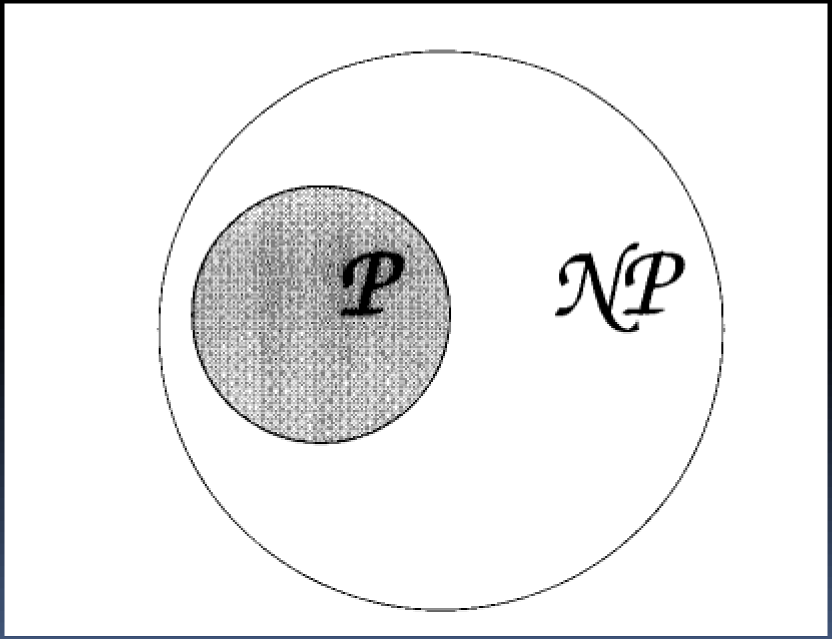
P and NP Problems

- *An algorithm A is of polynomial complexity if there exist a polynomial p( ) such that the computing time of A is O(p(n)).*
- An decision problem is one with yes/no answer
- **Class P:**
  - P is a set of all decision problems solvable by a deterministic algorithm in polynomial time.
  - **Examples:** Fractional Knapsack , Minimum Spanning Tree , Sorting problems (Bubble, Merge).
- **Class NP :**
  - NP stands for "Nondeterministic Polynomial-time"
  - NP is the set of all decision problems solvable by a nondeterministic algorithm in polynomial time.
  - Solved by Non-Deterministic Machine in polynomial time.
  - **Examples:** 0/1 Knapsack, Traveling Salesman, Graph Coloring, Satisfiability (SAT) problems

Relationship between P and NP

- Since deterministic algorithms are just a special case of nondeterministic ones, we conclude that

$$\mathcal{P} \subseteq \mathcal{NP}.$$



© 2021, S.Vineela Krishna, CSIT, CVR

NP Hard and NP Complete Problems

© 2021, S.Vineela Krishna, CSIT, CVR

Satisfiability (SAT)

- A formula ϕ is ***satisfiable*** if there exists an assignment of values to its variables that makes ϕ true.
- Let **x1, x2,…., xn** denote **boolean variables** whose value is either true or false.
- Let **E** be propositional formula.
- The satisfiability problem is to **determine whether a formula is true for some assignment of values to the variables (x1,x2,…xn).**
- **E(x1,x2,….xn) = true**

© 2021, S.Vineela Krishna, CSIT, CVR

Example Non Deterministic SAT algorithm

Algorithm Eval(E, n)
// Determine whether the propositional formula E is
// satisfiable. The variables are $x_1, x_2, \ldots, x_n$.
{
    for $i := 1$ **to** n **do** // Choose a truth value assignment.
        $x_i :=$ Choice(**false**, **true**);
    if $E(x_1, \ldots, x_n)$ **then** Success();
    else Failure();
}

© 2021, S.Vineela Krishna, CSIT, CVR

Reducibility

- A lot of times we can solve a problem by reducing it to a different problem.
- We can reduce Problem B to Problem A if, given a solution to Problem A, we easily construct a solution to Problem B
- Let L_1 and L_2 be problems.
- L_1 reduces to L_2 ($L_1 \alpha L_2$) iff there is a way to solve L_1 by deterministic polynomial time algorithm that solve L_2 in polynomial time.
- If we have a polynomial time algorithm for L_2 then we can solve L_1 in polynomial time.

© 2021, S.Vineela Krishna, CSIT, CVR

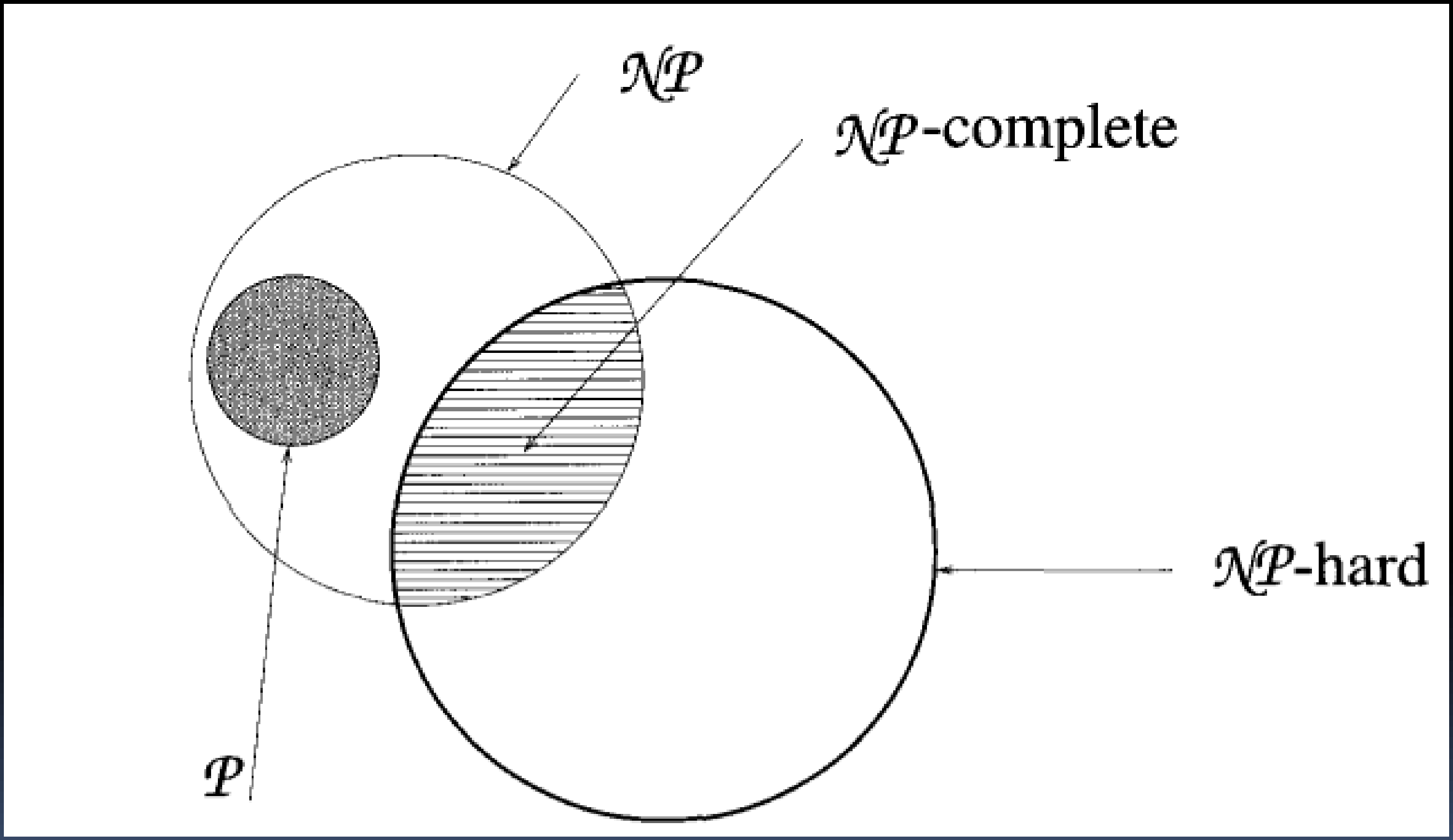
NP-Hard Problems:

- To prove a problem NP hard we need to reduce it to a problem which is already labeled NP hard.
- This reduction has to take polynomial time
- **"A Problem X is NP-Hard if there is an NP-Hard problem Y, such that Y is reducible to X in polynomial time. "**
- Base NP Hard problem which can be used for reduction is **Satisfiability problem**
- **"A Problem L is NP-Hard if and only if Satisfiability reduces to L (Satisfiability α L)."**
- NP-Hard Problem need not be in NP class.
- **Examples:**
  - The Circuit-satisfiability problem
  - Subset sum Problem
  - Travelling Salesman Problem

© 2021, S.Vineela Krishna, CSIT, CVR

- **NP-Complete problems:**
  - **"A problem L is NP-Complete if and only if L ∈ NP and L is NP-Hard." -** it is a part of both NP and NP-Hard Problems.
  - **It is the set of all decision problems whose solutions can be verified in polynomial time;**
  - A Problem **X** is ***NP-complete*** if it satisfies two conditions:
    - **X is in NP**
    - **Every problem in NP is reducible to X in polynomial time.**
  - A non-deterministic Turing machine can solve NP-Complete problem in polynomial time.
  - **Examples:**
    - **Knapsack problem**
    - **Graph coloring problem**
    - **Determining whether a graph has a Hamiltonian cycle**

Relationship between P, NP, NP-Hard and NP-Complete problems



© 2021, S.Vineela Krishna, CSIT, CVR

COOK's THEOREM

- *Cook in 1973 proved that the* ***Satisfiability problem(SAT) is NP-complete.***
- Cook's theorem, states that the "**Boolean Satisfiability problem is NP-complete**".
- That is, **1) it is in NP**, and **2) any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the Boolean Satisfiability problem**.

© 2021, S.Vineela Krishna, CSIT, CVR