* Nested Classes :→ (Inner Classes)

→ Anything dedicated only to that particular class only is written in nested classes.

* Can we create an object of an Interface?

```
interface Demo {
    void function();
}
class ABC implements Demo{
    defined here ;
}
```

→ Demo obj = new Demo();

Lambda Expression → Possible

* Sieve of Eratosthenes → Primes b/w 1 to N (1 & Itself)

i = 3 (true)

3×1 3×2 3×3

2 ✓ 4 12 6 14 8

j = i×i = 3×3 = (9)

~~3×5~~

(i×i) for Loop

12

PlayWorkSheet: Play & Learn

Table of Numbers From 1 to 100

1	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~	~~7~~	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	77	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
91	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

© playworksheet.com

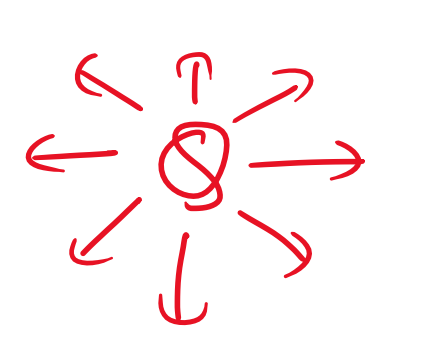
Introduction to Backtracking :→ (Application of Recursion)

* In certain problems we need to presume some conditions & move forward by blind guessing (Trial & Error) method.

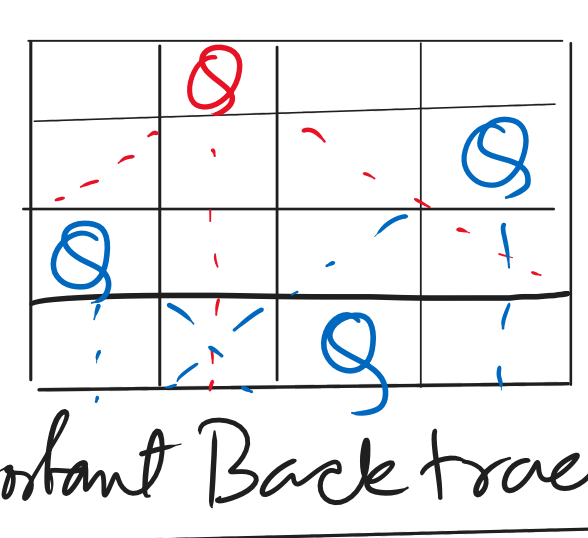
* When we move forward & don't find a solution, our initial prediction goes wrong & we have to revert back.

* This process of reverting back is called the Backtracking Algorithm & uses Recursion.

N Queens (4×4 Matrix)



4 Queens



Optimal solⁿ [4×4]

Important Backtracking Problems:

1. Rat in a Maze ××××
2. Sudoku Solver
3. Subsets of an Array (Power Set) / " " " String
4. Letter Combination of a Keypad (Phone keypad problem) (LC – 17)

(Rat In A Maze) (n×n) matrix

Dark Cells → Walls → 0

Green Cells → Safe Path → 1

Move forward → x+1

Move downward → y+1

(x,y) == 1

* curr { forward or downward } 1 0

All zeroes — Res Mat

LeetCode → (78) : Set of all subsets of an array/string :→

n = 3 arr = [1,2,3] → Mostly asked in F2F (Dry Run)

$2^3 = 8$ {{1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}, { }}

Recursion Tree :→ {1,2,3} → i/p { } → o/p

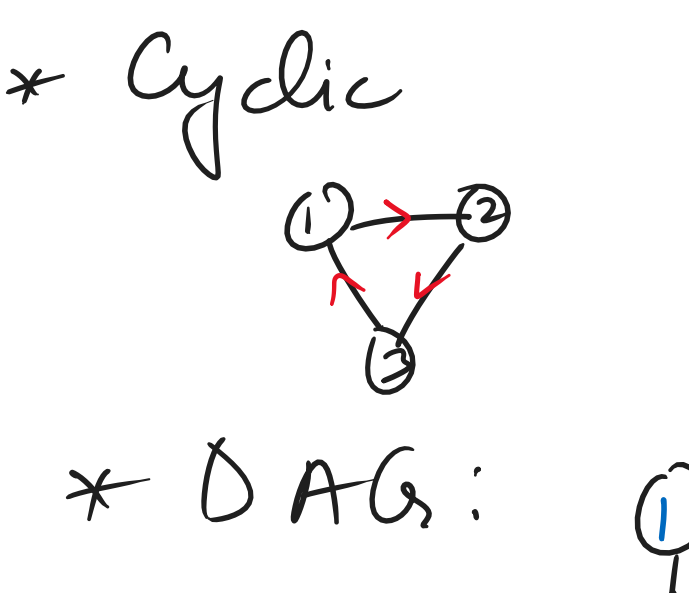
exclude: {1,2,3} { } include: {1,2,3} {1}

{1,2,3} { } {1,2,3} {2} {1,2,3} {1} {1,2,3} {1,2}

{1,2,3} { } (1) {1,2,3} {3} (2) {1,2,3} {2} (3) {1,2,3} {2,3} (4) {1,2,3} {1} (5) {1,2,3} {1,3} (6) {1,2,3} {1,2} (7) {1,2,3} {1,2,3} (8)

Introduction to Graphs :→ (DAG) ⇒ Directed Acyclic Graph

* Definition :→ A graph is a non-linear data structure which has two key entities connected in the diagram → Nodes & Edges



* Based on the connectivity there are many types of graphs.

(1) Undirected → Weighted, Unweighted

(2) Directed → Weighted, Unweighted

Weight :→ If an edge has some value, it is called the weight of that edge.

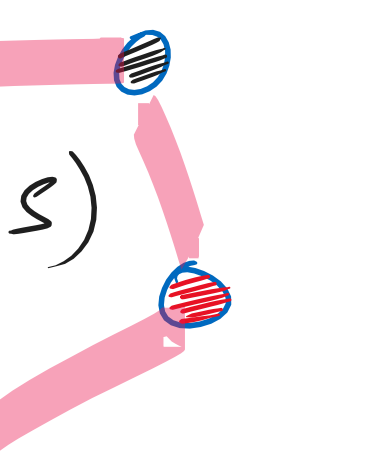
* Cyclic

* Acyclic

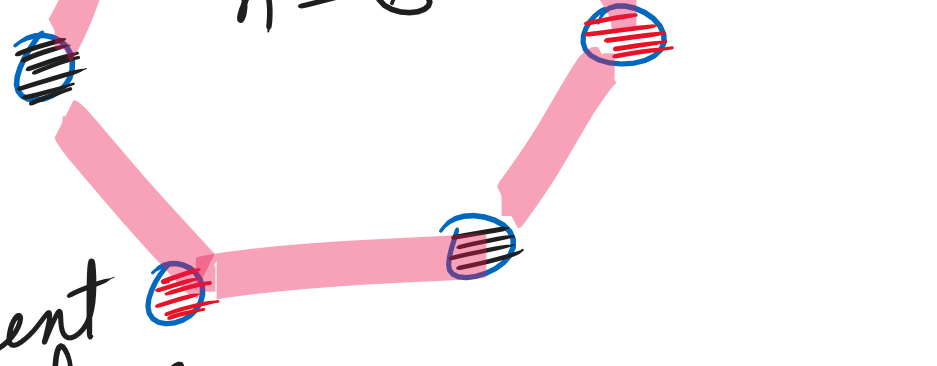
* Connected

Comp 1

* DAG:

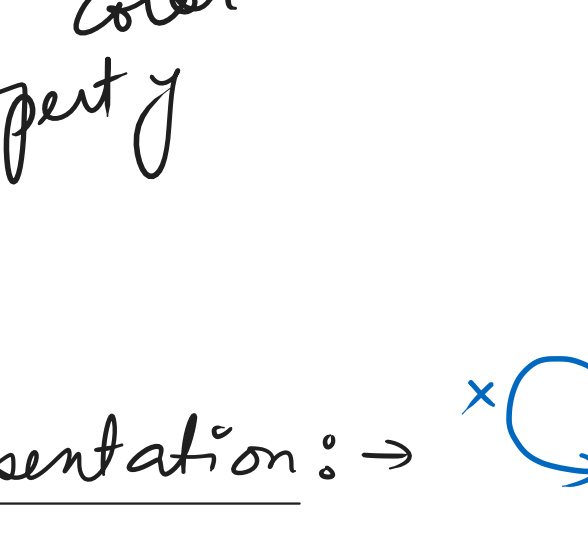


* Disconnected



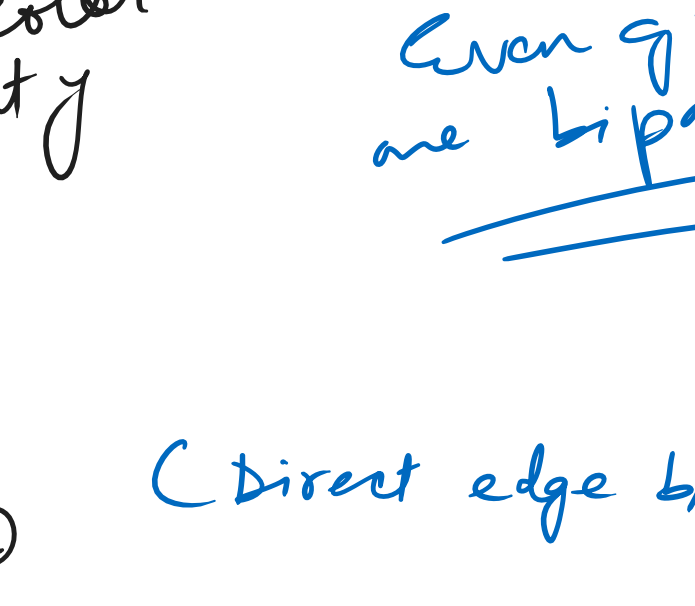
* Bipartite Graph:

Non-Bipartite



Adjacent Nodes have same colour property

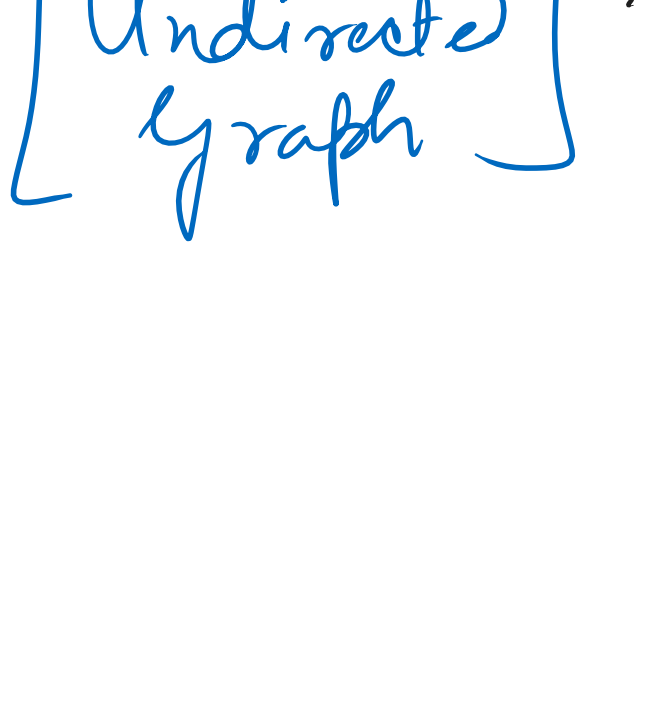
(Bipartite)



Adjacent Nodes have different colour property

Even graphs are bipartite

Representation :→



(Direct edge b/w u–v)

[Undirected graph]

* Adjacency Matrix

	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

u ⟷ v

(Pointers) DMA

* Adjacency List

Node : i	List of Neighbours
{1}	→ {2, 3, 4}
{2}	→ {1, 4}
3	→ 1, 4, 5
4	→ 1, 2, 3, 5
5	→ 3, 4

<List<List<Int>>