Graph
↳ Non-linear DS
↳ Dynamic
↳ as on other DS.
can be stored in type of graph.

60 - 70%
(Trees)
20 - 20 %
(Graphs)
20 - 10%
(Arrays)
(Linked list)
(Stack)

★ Graph doesn't have any fixed shape only.

Real world Example:
★ Google Maps

Undirected Graph

Directed Graph

Paths

Degrees

Degree(u)
↳ d(u) = 2
d(v) = 3

Directed
Indegree
Outdegree

Id (v) = 2.
od (v) = 2

Id (u) = 2
od (u) = 2

weight
(distance)
undirected weighted graph

Distance
u → v
= 3

Distance also a nodes is weight of the edge.

← Acyclic Graph

Cyclic Graph →

Types of Graphs

connected Graph

Trivial Graph

Regular Graph

Disconnected Graph

Acyclic Graph

Cyclic Graph

Graph Representation

① Maths → G(V, E)
vertex (Nodes) Edges

G(5,8)

G(5,6)

② Adjacency Matrix
③ Adjacency list

Adjacency Matrix

n×n
(square)
n = node

3×3 Matrix

	1	2	3
0	0	1	1
1	1	0	1
2	1	1	0

undirected Graph

n×n

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

Adjlist.

0 → 1, 2
1 → 0, 2
2 → 0, 1

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

Nodes neighbour (Count)

Size (Number of edges)

Author Creator ⟨id⟩ → adjlist. (Car)
Author.firstname ⟨id⟩ → adjlist (Car)

or

Hashmaps

{ 0: [1, 4, 5], ... }
key value

Graph Traversal Algos:-

★ BFS → Breadth first search (Level)
★ DFS → Depth first search ↳ Preorder, Inorder, Postorder
eg → (Tree) TLR LDR LRD

BFS

DFS

BFS

0
2
4
1
3

0 → 1, 2

DFS

0
1
2
3

Algo BFS

Req
★ adjlist ✓
★ Queue ✓
★ visited Array ✓
★ Ans Array

Adj
0 → 1, 2
1 → 2, 3, 4
2 → 0, 1, 3, 5
3 → 1, 2, 4, 5
4 → 1, 3
5 → 2, 3

Queue: 0 | 1, 2 | 3, 5

Answer

Initial Queue

0	1	2	3	4	5	6
X	X	X	X	X	X	

Visited

0	1	2	3	4	5	6
X	X	X	X	X	X	

f(6) → f(3) + f(5)
f(3) → f(1) + f(2)
f(5) → f(2) + f(4)

0, 1, 1, 2, 3, 5, 8

Algo DFS

Req → adjlist
★ visited

Adjlist:
0 → 1, 2
1 → 0, 3, 4
2 → 0, 5
3 → 1, 4
4 → 1, 3
5 → 2

visited
0 → F
1 → F
2 → F
3 → F
4 → F
5 → F
6 → T
5 → T
2 → T
0 → T

DP (Dynamic Programming)

★ whenever you make use past one results to repeat it.

Fibo → 0 1 1 2 3 5 8 ....
(0) (1) (1) (1) (2) (3) (5)

f(5) → 5

f(5) = f(4) + f(3)

f(4) = f(3) + f(2) → 2
f(3) = f(2) + f(1)

f(2) = f(1) + f(0) → 1
f(1) → 1
f(0) → 0

DP

★ Memoization → Create dp array then all the values are in the table
★ Tabulation → Build your program from bottom to up using the base cases

f(0) f(1) f(2)
↓ ↓ ↓
0 1 2

← dp array

0	1	2	3	4	5	6
0	1	1	2	3	5	8