Name Of Algorithm	Running Time
Depth First Search	O(V + E)
Breadth First Search	O(V + E)
Topological Sort	Θ(V + E)
Kruskal's Algorithm	O(ElgV)
Prim's Algorithm	O(E + VlgV)
Fractional Knapsack	O(nlgn)
Dijkstra's Algorithm	Worst = O(V^2) If min priority queue used (V+ElogV)
Longest Common Subsequence	

Graphs:

A graph is a data structure that consists of the following two components:
1. A finite set of vertices also called as nodes.
2. A finite set of ordered pairs of the form (u, v) called an edge.
The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost

Sparse graph: E is much less than V
Dense graph : Ei s close to V

Cyclic: A graph is cyclic if the graph comprises a path that starts from a vertex and ends at the same vertex. That path is called a cycle. An acyclic graph is a graph that has no cycle.

A tree is an undirected graph in which any two vertices are connected by only one path.

The most commonly used representations of a graph.
1. Adjacency Matrix (better use in dense graph)
2. Adjacency List (better use in sparse graph)

Adjacency List:

- An adjacency list is a collection of unordered lists used to represent a finite graph.The list represents the edges of the graph.
- An array of lists is used. The size of the array is equal to the number of vertices.
- The **space complexity** of adjacency list is O(V + E) because in an adjacency list information is stored only for those edges that actually exist in the graph
- For sparse graphs works better
- **Disadvantage:** No quicker way to determine whether a given edge is present in the graph than to search for v in the adj list.

Adjacency Matrix:

- Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
- Adjacency matrix for undirected graph is always symmetric
- **Space complexity** of the adjacency matrix is O(V2 independent of the number of edges.
- **Advantage :**Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).
- **Disadvantage:**Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.

BFS VS DFS

Sr. No.	Key	BFS	DFS
1	Definition	BFS, stands for Breadth First Search.	DFS, stands for Depth First Search.
2	Data structure	BFS uses Queue to find the shortest path.	DFS uses Stack to find the shortest path.
3	Source	BFS is better when target is closer to Source.	DFS is better when target is far from source.
4	Suitablity for decision tree	As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games.	DFS is more suitable for decision tree. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.
5	Speed	BFS is slower than DFS.	DFS is faster than BFS.
6	Time Complexity	Time Complexity of BFS = O(V+E) where V is vertices and E is edges.	Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.

Advantages of BFS:-

1. If Solution exists BFS will definitely find it
2. If 2 solutions exist the shortest one is found first
3. Never get trapped in unwanted nodes without the solution
4. Finds the closest goal in the least time

Disadvantages Of BFS :-

1. Each level of nodes are stored before creating the next one, hence it uses more memory
2. If the solution is far away from the root, BFS will consume more time

- BFS expands all children of a vertex and keeps them in memory. Then, it goes on to the next level of the first child

and expands its children and keeps them in memory as well. Hence, BFS keeps the one-hop fringe of all visited vertices in memory — which can lead to high memory usage.

- To avoid processing a node more than once, we use a boolean visited array.

Application Of BFS :

1.Finding Shortest Path.
2.Checking graph with bipartiteness.
3.Copying cheiney's Algorithm.
4.Torrent neighbor tracking

Procedure:
1. Source Declared : Null the parent
:Distance infinity
: White all
2. Null the source
   - Distance zero
   - Visited Grey
3. Source inserted in Queue
4. Source check child
5.Child Enqued
6. Source Black

>BFS uses first in first out queue
>Shortest path is the minimum number of edges taken.

Advantages Of DFS :-

1. Find largest distance elements in least time
2. Less memory is required since its linear with respect to nodes
3. It is well suited if it has goals in all paths, Eg., If we search for Starbucks from my location, it will definitely find one soon. but if we search for a museum, then it will take more time and might not guarantee to find one

Disadvantage of DFS :-

1. Complexity depends on the number of paths
2. Cannot check duplicate nodes
3. Get trapped in searching useless paths without a solution
4. Cannot guarantee to find a solution
5. Cannot find the minimal solution if two solutions are available

Applications of DFS:-

1. Finding Connected components.
2. Topological sorting.
3. Finding Bridges of graphs.

>DFS Visit is a recursive function.
>DFS timestamps each vertex
>DFS structure exactly mirrors the structure of recursive calls of DFS visit.
>DFS follows parenthesis structure in terms of starting and finishing time.

- Predecessors subgraph a shortest (in terms of weight) path tree in the graph, which joins all the vertices which have a predecessor.

Parenthesis Theorem:

> If v is a descendant of u, then the discovery time of v is later than the discovery time of u.

In any DFS traversal of a graph g = (V, E), for any two vertices u and v exactly one of the following holds:

- The intervals [d[u], f[u]] and [d[v], f[v]] are entirely disjoint and u nor v is a descendant of the other in the depth-first forest.
- The interval [d[u], f[u]] is contained within the interval [d[v], f[v]], and u is a descendant of v in a depth-first tree.
- The interval [d[u], f[u]] is contained entirely within the interval [d[v], f[v]], and v is a descendant of u in a depth-first tree.

Q.Why the predecessor subgraph of a depth-first search forms a depth-first forest not a tree? _Because we cannot assume every vertex can be reached starting from the source vertex chosen, in which case we have to pick another source(thus another root node in the forest) and continue.

Topological Sort:
A topological sorting of DAG is a linear ordering of all its vertices.

>DAG : Directed Acyclic Graph
>Start from the node that has no dependency /no incoming edge
> No linear ordering is possible if graph contains a cycle

Procedure:
1. Do DFS to computer finishing time
2.As each vertex is finished, insert it to the linked list.
3. Sort according to finishing time. Return the linked list of vertices.

Greedy Algorithm :
GA works by making decisions that seem promising at any moment. (Short sighted in their approach) It makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. It's faster to execute .

Disadvantage:
>Does not always produce an optimal solution to every problem
>Correctness hard to prove

- Greedy choice property: A global (overall) optimal solution can be reached by choosing the optimal choice at each step.
- Optimal substructure: A problem has an optimal substructure if an optimal solution to the entire problem contains the optimal solutions to the sub-problems.

>A greedy algorithm is similar to a dynamic programming algorithm. Only difference is that solutions to subproblems do not have to be known at each stage.

>Kruskal and Prim's algorithms are greedy algorithms , these are used to find the minimum spanning trees.

Minimum Spanning Trees:
A minimum spanning tree or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

>a spanning tree does not have cycles and it cannot be disconnected.

Kruskal's Algorithm
builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows a greedy approach as in each iteration it finds an edge which has least weight and adds it to the growing spanning tree.
Algorithm Steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which don't form a cycle , edges which connect only disconnected components.

Running time : O(Elog V)

Prim's Algorithm
also uses the Greedy approach to find the minimum spanning tree. In Prim's Algorithm we grow the spanning tree from a starting position. Unlike an edge in Kruskal's, we add vertex to the growing spanning tree in Prim's.
Algorithm Steps:

- Maintain two disjoint sets of vertices. One containing vertices that are in the growing spanning tree and other that are not in the growing spanning tree.
- Select the cheapest vertex that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using Priority Queues. Insert the vertices that are connected to the growing spanning tree, into the Priority Queue.
- Check for cycles. To do that, mark the nodes which have been already selected and insert only those nodes in the Priority Queue that are not marked.

Running time : O(E + VlogV)

Disjoint sets are sets whose intersection is the empty set so it means that they don't have any element in common.

Single Source Shortest Path
Can be used to solve these :

- Single destination shortest Problem
- Reverse all edges
- Start from destination(single) source
- Reverse again
- Single Pair shortest Problem
- Direction matters, (u,v) Find single source shortest path u to v. This will be a single pair shortest path.
- All pair shortest Problem
    - Start from all vertex
    - Gives all pair possible

- Optimal substructure property: If a path is shortest (true) It's subpath will be shortest also.

// Initialization:
:All vertex distances will be infinity.
Infinity means that it can't be reached or has not been explored yet.

: Source distance is made zero. No cost to go from source to source
: Relaxation _ Checks whether to take this path or not (improvements)
Decreases the value of the shortest path.
:Update when less than existing cost.

Dijkstra's Algorithm :

is an algorithm for finding the shortest paths between nodes in a graph.

> Directed weighted graph
>No negative edges
>Maintains a set “S” for vertices whose shortest path are already known

We need to maintain the path distance of every vertex. We can store that in an array of size v, where v is the number of vertices.
We also want to be able to get the shortest path, not only know the length of the shortest path. For this, we map each vertex to the vertex that last updated its path length.
Once the algorithm is over, we can backtrack from the destination vertex to the source vertex to find the path.
A minimum priority queue can be used to efficiently receive the vertex with least path distance.
Time Complexity: `O(E Log V)`

Divide and Conquer	**Dynamic Programing**
● Solve problems by dividing problems into subproblems ● Subproblems are not dependent ● Does not store solution of subproblems ● Top down algorithm ● Example: merger, quick, binary sort	● Solve problems by dividing problems into subproblems ● Subproblems are dependent ● Store solution of subproblems ● Bottom up algorithm ● Example: Knapsack, Matrix chain multiplication

Dynamic Programming:
We can apply DP only in two cases:

- If the problem has optimal substructure Property
- If the problem has overlapping subproblems

It is Recursion + Memoization. It helps to improve running time of O(2^n) to O(n)

Longest Common Subsequence:

Any string with length N will have 2^n number of subsequences. Subsequence of ABC is : A,B,C,AB,BC,AC,ABC, and Φ

The longest common subsequence (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences)

For subsequence:
> Sequence matters
> Can;t be reversed
> Order has to be maintained
>Contiguous doesn't matter
